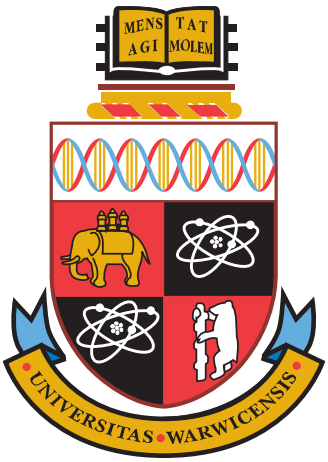


High-fidelity Rendering on Shared Computational Resources

Vibhor Aggarwal

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Engineering

School of Engineering
University of Warwick
October 2010

Contents

List of Figures

List of Tables

Acknowledgements

This thesis would not have been possible, had Alan not replied to one of the many random emails he gets daily and if Kurt had not instigated me into the intriguing world of research during my undergraduate internship. It is only fitting that both of them jointly supervised me on this thesis. I am highly indebted to them for their constant support and encouragement while enthusiastically motivating me through this journey.

The Visualisation Group at the University of Warwick has been immensely helpful in providing a strong sense of a research community. Both Tom and Piotr provided invaluable assistance during the research, offering useful insights and advice. Vedad and Jass were always present whenever I got stuck. Alena, Alessandro, Belma, Carlo, Ela, Elmedin, Francesco, Gabriela , Jasminka, Matt, Mike, Peter, Remi, Sandro, Silvester, Simon: it was great working with all of you and I especially enjoyed our outings together.

Usama, who tragically passed away in October 2007, deserves a special mention. He once showed me how he could do "magic" by reconstructing images from sparse samples and thus inspired me to use these techniques for my research. His ever-smiling face will always be an inspiration.

The thesis writing group: Claire, Jayne, Kim, Rosario and Shaobo, had a crucial role in getting this thesis written. They were always there when I struggled to put words on the paper.

My wonderful experience at Warwick has been enriched by countless number of friends I made here. They have been responsible for providing a truly international environment, exposing me to the multitude of cultures from around the world. Ankit, David, Hardik, Himanshu, Hitesh, Katty, Kavit, Marvin, Marzia, Max, Merve, Mouz, Nagesh, Nur, Prakash, Ramina, Saurin: being so far away from home would not have been so easy without you.

Finally, my parents and family have always supported me tirelessly in all my endeavours. I could not have done this without your affection and kind thoughts!

I would like to express my gratitude to all the people, not only just those mentioned above, but numerous others as well who were part of the whole PhD process and assisted me in various ways during the past three years. I am also thankful to the University of Warwick for funding my research through the Vice Chancellor's scholarship.

Declaration

The work in this thesis is original and no portion of work referred to here has been submitted in support of an application for another degree or qualification of this or any other university or institute of learning.

Signed: Date: 15 October 2010

Vibhor Aggarwal

List of Publications

The following have been published as a result of the work contained within this thesis.

Journal paper

- **Aggarwal V.**, Debattista K., Bashford-Rogers T., Dubla P., Chalmers A.: *High-fidelity Interactive Rendering on Desktop Grids.* IEEE Computer Graphics and Applications, PrePrints (2010).

Peer-reviewed Conference Papers

- **Aggarwal V.**, Chalmers A., Debattista K.: *High-Fidelity Rendering of Animations on the Grid: A Case Study.* In Eurographics Symposium on Parallel Graphics and Visualization (Crete, Greece, 2008), Eurographics Association.

- **Aggarwal V.**, Debattista K., Dubla P., Bashford-Rogers T., Chalmers A.: *Time-constrained High-fidelity Rendering on Local Desktop Grids.* In Eurographics Symposium on Parallel Graphics and Visualization (Munich, Germany, 2009), Eurographics Association.

Peer-reviewed Poster

- **Aggarwal V.**, Debattista K., Chalmers A.: *High-fidelity Rendering using Distributively-controlled Multi-programmed Computing.* In TCPP PhD Forum, IEEE International Parallel and Distributed Processing Symposium (Rome, Italy, 2009), IEEE Computer Society.

Abstract

The generation of high-fidelity imagery is a computationally expensive process and parallel computing has been traditionally employed to alleviate this cost. However, traditional parallel rendering has been restricted to expensive shared memory or dedicated distributed processors. In contrast, parallel computing on shared resources such as a computational or a desktop grid, offers a low cost alternative. But, the prevalent rendering systems are currently incapable of seamlessly handling such shared resources as they suffer from high latencies, restricted bandwidth and volatility. A conventional approach of rescheduling failed jobs in a volatile environment inhibits performance by using redundant computations. Instead, clever task subdivision along with image reconstruction techniques provides an unrestrictive fault-tolerance mechanism, which is highly suitable for high-fidelity rendering. This thesis presents novel fault-tolerant parallel rendering algorithms for effectively tapping the enormous inexpensive computational power provided by shared resources.

A first of its kind system for fully dynamic high-fidelity interactive rendering on idle resources is presented which is key for providing an immediate feedback to the changes made by a user. The system achieves interactivity by monitoring and adapting computations according to run-time variations in the computational power and employs a spatio-temporal image reconstruction technique for enhancing the visual fidelity. Furthermore, algorithms described for time-constrained offline rendering of still images and animation sequences, make it possible to deliver the results in a user-defined limit. These novel methods enable the employment of variable resources in deadline-driven environments.

CHAPTER 1

Introduction

In the past few decades, many researchers have been attracted to the field of high-fidelity rendering in search of techniques which can generate realistic images of virtual environments quickly and accurately, by exploiting the advances made in modern computing architectures. Recently, computational platforms have emerged which are designed for sharing resources between the users. These platforms allow their users to benefit from increased computational power at a reduced cost by aggregation of resources. This thesis is the first to investigate the challenges for high-fidelity rendering on such shared computational resources, and presents novel algorithms to tackle them.

1.1 High-fidelity Rendering

Rendering is the process of digital generation of imagery using the description of a virtual scene containing information about its geometry, lighting, material properties and camera attributes. This can be classified into two main categories: photorealistic and non-photorealistic rendering. The former deals with generation of images which are similar to a real photograph of the scene while the latter imitates artistic representations such as paintings and cartoons.

High-fidelity rendering uses physically-based quantities for generating photorealistic images and is the focus of this thesis, see for example Figure 1.1. This involves computation of light transport through a virtual scene by simulating light interactions between the surfaces present in it, thereby determining the amount of light reaching the camera. A global illumination lighting model is used such that light interactions between all the surfaces are taken into account for the image generation process, resulting in effects which occur in real world

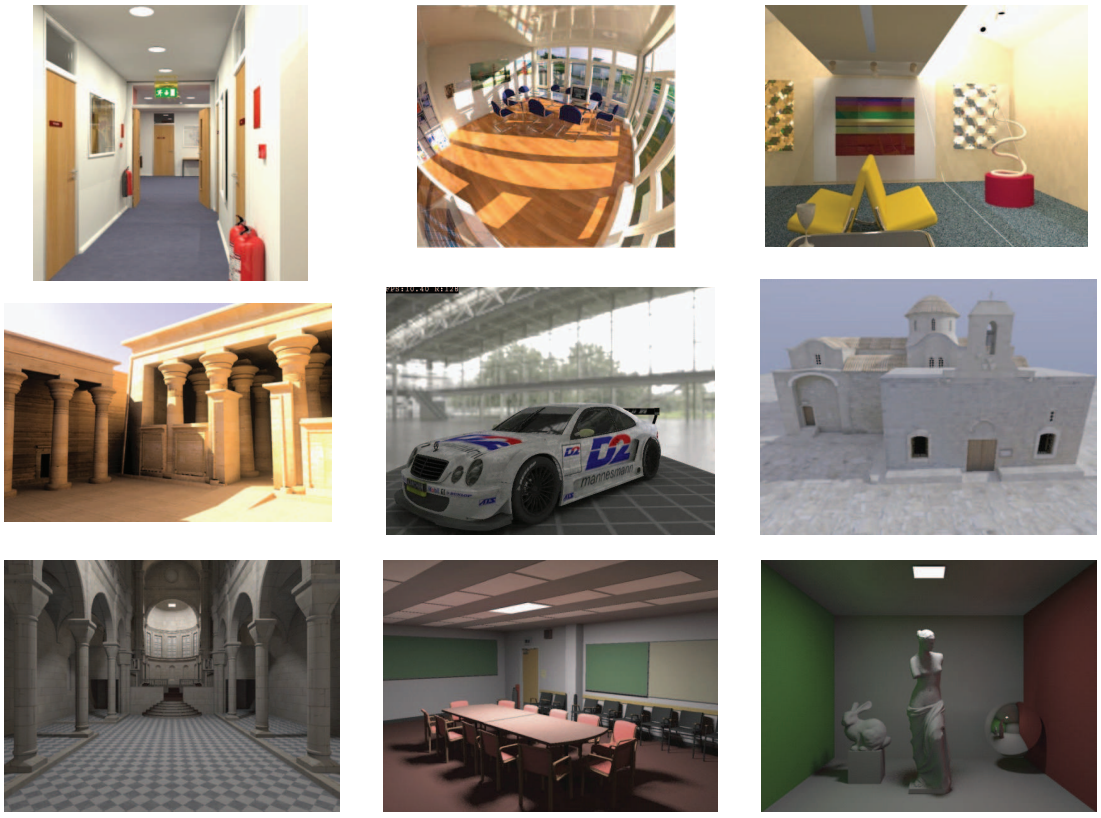


Figure 1.1: High-fidelity rendering examples

such as colour bleeding, soft shadows, caustics etc. A mathematical model for the whole process was formulated by Kajiya [Kaj86], known as the rendering equation. The major challenge lies in solving this equation, due to the associated computational expense.

1.1.1 Applications

The application of high-fidelity rendering techniques has been restricted due to their computationally expensive nature. However, the advancements in the rendering techniques and computer hardware, including specialised Graphics Processing Units (GPUs), have increased their scope to a great extent as evident from the following examples [DBB06]:

- Architecture: Architects employ high-fidelity rendering to generate realistic images and walk-throughs of building designs [BHWL99, ASKCK03], to help their clients visualise them under different interior and exterior lighting conditions. These renderings enable the architects to study the lighting

conditions and modify the designs according to the client's preferences before the construction starts.

- Archaeology: Many archaeological sites around the world have been damaged with time and archaeologists reconstruct them to be able to preserve and study cultural heritage [HMD*10]. High-fidelity renderings of virtual reconstructions assist them to view sections of the sites which may not exist any more. Relighting the cultural heritage sites with authentic lighting conditions helps them recreate the past with a greater accuracy.

- Visual Effects: The modern movie industry relies heavily on computer generated imagery [TL04,PFHA10]. This allows them to add virtual characters into a scene or enhance the visual appeal of the scene using special effects. For these additions to be believable, the lighting conditions used for virtual components need to be similar to that of the real scene and high-fidelity rendering provides this facility.

- Lighting Design: High-fidelity rendering techniques provide a practical tool for lighting engineers [GGHS03]. It helps them to design and study light sources and characterise their emission properties by placing them in virtual environments. They can then verify these findings by conducting physical measurements.

- Computer Games: For the past few years, the computer games industry has been constantly pushing the limits of the current computer graphics capabilities to offer gamers an immersive environment by generating ever more realistic images [Mit07]. Computer games these days are increasingly using high-fidelity techniques for this, however, due to the computationally expensive nature of these techniques, they have been usually limited to pre-rendered visuals.

- Product Design: Product designers also benefit from high-fidelity renderings. This provides them with a useful tool for enhancing their designs of products such as cars, furniture, appliances, consumer electronics by simulating their appearance under various lighting conditions [RMS*08,KLN09]. Furthermore, such images can also be used for advertising new products even before they have been produced.

- Training Simulators: An accurate visual stimulus enhances the effectiveness of training simulators as it provides the users with a good representation of real environments, preparing them for scenarios they are likely to encounter in real life. For example, generating precise visuals for different visibility levels for use in a flight simulator is very important to train the pilots for adverse weather conditions [Nie03].

High-fidelity rendering allows these applications to study and visualise real-world scenarios at a much lower cost in a safe and controlled manner. For example, physical construction of a building would require much more money and time than creating a virtual model. Also, modification of a virtual prototype is a lot more simple than changing a physical entity. Furthermore, it may not always be feasible to create a real-world alternative. For example, replacement of virtual characters used in movies and recreation of scenarios used for training simulators. Hence, high-fidelity rendering is crucial for these applications and any future advancements made in rendering would be beneficial for them.

1.2 Shared Computing Resources

The advances made by the semiconductor industry following Moore's Law have led to a considerable increase in computational power of modern day Central Processing Units (CPUs). This has resulted in a widespread use of computers to solve a multitude of challenges faced by science, engineering and businesses. However, the computational demands of such applications have continuously exceeded the processing power of the current generation computers. Hence, parallel and distributed computing in various shapes and forms have been employed to address this issue.

Traditional parallel systems have been either shared memory machines (supercomputers) or distributed multi-processors connected by high speed networks (clusters). The high cost of installing and maintaining such systems has restricted their user base. There is a growing class of applications which process information from a combination of heterogeneous components and they are usually unsupported by such traditional parallel systems. Researchers have been looking to overcome these problems of cost and heterogeneity by creating mechanisms for aggregating shared distributed resources across multiple domains. This is known as grid computing, and the term was coined analogous to the electrical power

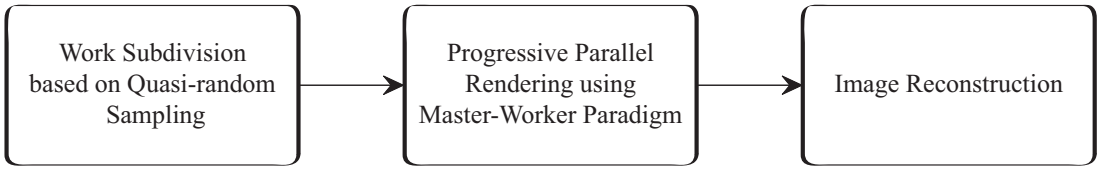


Figure 1.2: A novel fault-tolerant mechanism for high-fidelity rendering which forms the basis of many algorithms presented in this thesis.

grid [FK99]. Grid computing is typically employed to tackle large problems that can not be solved using resources owned by a single user.

Grid computing inherits the obstacles faced by distributed computing such as fault-tolerance, synchronisation, communication, heterogeneity, load balancing and scheduling. In addition to these challenges, grid computing needs to tackle effective sharing and management of distributed resources for a seamless user experience.

1.3 Rendering on Shared Resources

The computationally expensive nature of high-fidelity rendering has led the researchers to employ parallel computing to alleviate this cost. Many parallel rendering algorithms have been devised exploiting either data parallelism by decomposing the computations in object space or task parallelism by decomposing them in image space [CDR02]. The traditional parallel rendering algorithms are designed for supercomputers or clusters known as render farms and rely on quick and frequent communication between the rendering processes. These render farms are prohibitively expensive, typically costing in excess of £100,000, which restricts the scope of parallel rendering. Shared computational resources provide a much cheaper alternative by aggregating idle (already present) resources. But parallel rendering until now, has not been designed to be fault-tolerant and handle run-time variation of resources and hence it is incapable, in any traditional form, of effectively employing these shared resources.

The conventional fault-tolerance mechanisms for tackling unreliable systems employ checkpointing or replication or a hybrid of the two approaches. This results in an extra computational load which limits the performance of algorithms employing these fault-tolerance strategies. A tradeoff between fault-tolerance and performance is usually made for computing on shared resources. A novel fault-tolerance mechanism is presented in this thesis for high-fidelity rendering which

Figure 1.3: A sequence of frames rendered interactively, using the novel fault-tolerance mechanism on volatile resources.

employs quasi-random sequences and image reconstruction techniques without inhibiting performance, see Figure 1.2.

The variable computational power provided by shared resources makes it difficult to employ them in a deadline-driven environment. The use of time-constraints to restrict the computational time is an elegant way of employing these volatile resources in a production environment where strict deadlines need to be met. A progressive rendering technique along with the novel fault-tolerant mechanism allows maximisation of the visual quality of the generated imagery within a given time limit.

High-fidelity rendering at interactive rates offers the potential for high qual-

ity imagery to be used as a regular tool in many visualisation applications for providing an immediate response to the changes made by the user. This would allow the users to test various parameters of a scene, which is challenging while rendering in an offline environment due to the time involved in such a process. Interactive rendering requires high-performance computing due to the rate at which the imagery is calculated and updated. Shared computing resources have been traditionally used for high-throughput computing. The novel fault-tolerant mechanism proposed in this thesis enables them to act as a high-performance resource, as illustrated by the interactively rendered images shown in Figure 1.3.

The generation of high-fidelity animations is a time consuming process even on render farms, as all individual frames of an animation need to complete before they can be viewed. An animation studio usually renders multiple drafts of such animations before finalisation. Time-constrained rendering of such animations on idle resources of an institution would increase their utilisation, while decreasing the reliance on render farms. Enabling efficient utilisation of existing resources would help in reducing operational costs, avoiding the need for expensive render farms.

1.4 Research Objectives

The conjunction of high-fidelity rendering and shared computational resources has been, until now, mostly unexplored. This thesis aims to develop novel fault-tolerant rendering algorithms for taking advantage of inexpensive shared resources, building upon the existing knowledge in both the areas of high-fidelity rendering and shared computational resources. The main research objectives of this thesis are:

- to develop high-fidelity rendering algorithms with restricted communication to take advantage of massive computational power offered by computational grids.
- to devise a fault-tolerance mechanism for high-fidelity rendering which does not restrict performance unlike existing fault-tolerance strategies.
- to develop time-constrained offline rendering algorithms using the novel fault-tolerance mechanism to employ variable resources of a desktop grid.

- to create an interactive rendering system capable of handling run-time variations in computational power.
- to develop time-constrained animation rendering algorithms to employ desktop grids in a production environment.

1.5 Organisation

This thesis is organised as follows:

Chapter 2: High-fidelity Rendering presents an overview of high-fidelity rendering and describes the relevant previous work which has been carried out in this field.

Chapter 3: Computing on Shared Resources covers the background on grid computing and its relevance for high-fidelity rendering.

Chapter 4: Quasi-random Sequences The novel algorithms presented in this thesis rely heavily on quasi-random sequences and this chapter provides a practical guide on them.

Chapter 5: Animation Rendering on Computational Grids presents and compares two algorithms developed for rendering animations on computational grids and discusses the issues encountered while employing them. It shows how a two-pass approach can achieve speed-up and better visual fidelity than existing techniques.

Chapter 6: Time-constrained Offline Rendering on Desktop Grids describes a novel fault tolerant mechanism which uses quasi-random sampling and image reconstruction techniques. This does not inhibit performance unlike traditional fault-tolerance strategies. It employs this mechanism for time-constrained rendering on desktop grids and presents and compares two algorithms for offline rendering. The advantage of using a component-based approach, by task subdivision at finer granularity, for parallel rendering is demonstrated.

Chapter 7: Interactive Rendering on Desktop Grids imposes a stricter time limit on the parallel rendering system for achieving interactive rates

on desktop grids. A real-time spatio-temporal image reconstruction mechanism is presented which helps in achieving better visual fidelity in the case of sparse sampling. To the best of the author's knowledge, this is the first interactive desktop grid system that was developed by creating a new job scheduling architecture.

Chapter 8: Time-constrained Animation Rendering on Desktop Grids describes and compares two algorithms developed for rendering animation with a time-constraint. It shows that an approach based on multi-dimensional quasi-random sampling has a superior performance than traditional animation rendering techniques as it progressively refines the whole animation, even in the presence of faults.

Chapter 9: Conclusion and Future Work concludes the thesis and provides directions for potential future work.

CHAPTER 2

High-fidelity Rendering

This chapter presents a background on high-fidelity rendering concepts and algorithms while highlighting relevant previous research which has been carried out in this field. It begins by introducing radiometric quantities to derive the rendering equation. Subsequently, three approaches for solving the rendering equation: rasterisation, radiosity and ray tracing are discussed. Next, a description of rendering techniques used in this thesis is provided. This is followed by an overview of image reconstruction and sparse sampling. Finally, research work carried out in the areas of time-constrained rendering and parallel rendering is presented.

2.1 Radiometry

The generation of high-fidelity renderings requires computation of light energy in a scene. Radiometry is a field which studies the measurement of light energy and hence it is employed for high-fidelity rendering computations. Photometry is another related area of study which deals with perceived brightness of light to the human eye. The human visual system is not equally sensitive to all the wavelengths in the visible spectrum and photometric quantities take this into consideration. However, radiometric quantities are used for high-fidelity rendering and corresponding photometric quantities can be successively computed [DBB06]. A description of some radiometric quantities, which are important from the perspective of high-fidelity rendering, are presented below following the notations from [DBB06].

2.1.1 Radiometric Quantities

2.1.1.1 Radiant Power

This radiometric quantity is useful for estimating the rate of light energy emanating from a light source or reaching a surface. It is also known as flux (Φ) and is measured in watts, W (joules/sec). Radiant power is independent of the size of source or the surface or the distances between them. A common use of radiant power is to categorise the light sources, for example a light bulb labelled 100W transforms approximately 100J of energy every second.

2.1.1.2 Irradiance

Irradiance (E) denotes the radiant power per unit area received on a surface. It is measured in watts/m^2.

$$E = \frac{d\Phi}{dA}$$

2.1.1.3 Radiant Exitance/Radiosity

The amount of radiant power emanating per unit area is termed radiant exitance (M) or radiosity (B). It is also measured in watts/m^2. The need for defining two radiometric quantities to express radiant power per unit area arises since the incident and exitant light can be potentially different at the same point.

$$M = B = \frac{d\Phi}{dA}$$

2.1.1.4 Radiance

Radiance is used to measure the amount of light arriving at a point in a given direction. It is the radiant power per unit projected area per unit solid angle (watts/steradian·m^2). The projected area refers to the area of the surface perpendicular to the direction. The radiance, $L(x \rightarrow \Theta)$, at a point x in the direction Θ is given by (see Figure 2.1):

$$L(x \rightarrow \Theta) = \frac{d^2\Phi}{d\omega dA \cos\theta}$$

Radiance is the preferred radiometric quantity for generating high-fidelity images because it can be closely associated with what the eye senses, that is the

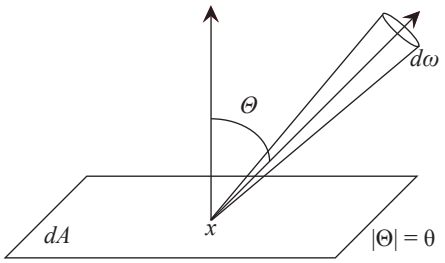


Figure 2.1: Description of quantities for calculating radiance, $L(x, \Theta)$, at a point x in direction Θ, after [DBB06]

appearance of an object [SM09, Jen01]. Furthermore, it is invariant across a line in space in a vacuum.

The relationships between the radiometric quantities described above can be defined as:

$$\Phi = \int_A \int_\Omega L(x \rightarrow \Theta) \cos\theta d\omega_\Theta dA_x$$

$$E(x) = \int_\Omega L(x \leftarrow \Theta) \cos\theta d\omega_\Theta$$

$$B(x) = \int_\Omega L(x \rightarrow \Theta) \cos\theta d\omega_\Theta \tag{2.1}$$

where Ω is the total solid angle and A is the total surface area. $L(x \rightarrow \Theta)$ represents the radiance leaving and $L(x \leftarrow \Theta)$ represents radiance reaching point x.

2.1.2 Bidirectional Reflectance Distribution Function

The characterisation of surface properties of the materials is needed while performing light transport computations in a scene. When light falls on a surface it may be absorbed, reflected or transmitted at the same or a different point from the original point of incidence. For example, materials such as marble and human skin exhibit subsurface scattering phenomenon, that is the light incident on them gets absorbed and scatters at different points. A Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF) is defined to model the light interaction properties of materials. A simpler function, Bidirectional Reflectance Distribution Function (BRDF), f_r, was defined by Nicodemus [Nic65] with the assumption that light is incident and reflected at the same point, ignoring subsurface scattering. BRDF represents the ratio between the radiance reflected in the direction Θ and the irradiance incident through an angle, ω_Ψ at a point x. It

is given by:

$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL_r(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} = \frac{dL_r(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x, \Psi)d\omega_\Psi} \tag{2.2}$$

where, N_x is the normal vector at x.

Many types of BRDFs have been proposed to model different types of materials such as diffuse, glossy and specular. Kurt and Edwards [KE09] present a survey of these.

2.2 The Rendering Equation

The equilibrium of light energy in a scene can be mathematically formulated, using the radiometric quantities defined in the previous section, to define the radiance at any point in that scene. This equation is termed as the rendering equation [Kaj86] and it forms the basis for high-fidelity rendering algorithms. It states that the outgoing radiance, $L(x \rightarrow \Theta)$ at a point is equal to the sum of the emitted radiance, $L_e(x \rightarrow \Theta)$ and the reflected radiance, $L_r(x \rightarrow \Theta)$ based on the conservation of energy.

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta) \tag{2.3}$$

Integrating Equation(2.2):

$$L_r(x \rightarrow \Theta) = \int_{\Omega_x} f_r(x, \Theta \leftrightarrow \Psi)L(x \leftarrow \Psi)\cos(N_x, \Psi)d\omega_\Psi \tag{2.4}$$

Substituting in Equation(2.3), to obtain the rendering equation:

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Theta \leftrightarrow \Psi)L(x \leftarrow \Psi)\cos(N_x, \Psi)d\omega_\Psi \tag{2.5}$$

This can be also transformed into an area formulation such that the radiance can be computed by integrating over all the surfaces of a scene represented by set A:

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_A f_r(x, \Theta \leftrightarrow \Psi)L(y \rightarrow -\Psi)V(x, y)G(x, y)dA_y \tag{2.6}$$

where,

$$G(x, y) = \frac{\cos(N_x, \Psi)\cos(N_y, -\Psi)}{r_{xy}^2} \tag{2.7}$$

and $V(x, y)$ represents the visibility term between two points x and y which is 1 if x and y are mutually visible and 0 if not. The above equations assume that the light travels instantaneously through a vacuum and there is no participating media within the scene. Furthermore, they are valid for a single wavelength only and do not account for phenomena such as diffraction, polarisation and interference. Modifications to the rendering equation for incorporating participating media, for example smoke and dust, can be found in [Jen01].

The rendering equation can be classified as second degree Fredholm integral equation since the quantity to be calculated, radiance, appears on the left hand side and on the right hand side as part of an integral and it cannot be solved analytically. There are two approaches to solve the rendering equation: Finite element methods and point sampling methods. The radiosity approach (see Section 2.4) is a finite element method, while rasterisation (see Section 2.3) and ray tracing based algorithms (see Section 2.5) are point sampling methods for estimating the rendering equation for a scene. The algorithms proposed in this thesis are suited for point sampling methods only.

2.3 Rasterisation

There are two major categories of digital image synthesis techniques: object-order and image-order [SM09]. Object-order methods iterate through each object in the scene for generating images while image-order methods iterate through each image pixel and find the objects which influence its colour. Rasterisation techniques are object-order methods which map the scene geometry to image pixels whereas ray tracing based algorithms are image-order methods. Rasterisation has been brought to the forefront in comparison to ray tracing methods, due to the support provided by GPUs which implement these techniques in hardware and APIs such as OpenGL and DirectX. Although lately, with the increased flexibility of GPU programming, ray tracing methods [PBD*10] have also been developed for exploiting the performance boost provided by the GPU.

Rasterisation has been the popular technique for generating interactive visuals due to its speed. However, as it does not natively handle global illumination effects, generation of realistic images is difficult. It approximates effects such as shadows, reflections and refractions which ray tracing techniques can calculate precisely. Furthermore, scene geometry needs to be tessellated for rasterisation

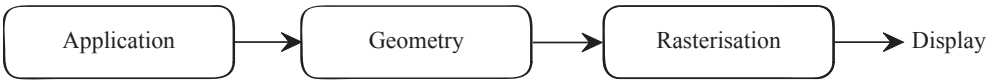


Figure 2.2: The graphics pipeline for rasterisation, after [AMHH09]

as it deals with polygons, in contrast to ray tracing which is not bound by such a requirement.

The graphics pipeline for generating images using rasterisation can be broadly classified into three stages [AMHH09]: application, geometry and rasterisation (see Figure 2.2), where each of these stages can be further decomposed. The application stage is responsible for the generation of primitives which need to be displayed. The geometry stage performs various operations on these primitives such as modelling transformations, per-vertex illumination, viewing transformations, clipping and projection before they can be rasterised. The process of rasterisation can be broken into three main subtasks [FvDFH97]: visible surface determination, scan conversion and shading. First, it identifies the polygons which are visible from the current camera view, then it determines the pixels which are covered by the visible polygon and finally it colours the pixels based on a shading algorithm. A plethora of techniques exist for accelerating rasterisation while increasing the degree of realism such as texture filtering, bump mapping, environment mapping, level of detail, shadow volumes etc. and an overview of these can be found in [AMHH09].

2.4 Radiosity

Radiosity methods provide a view-independent approach for computing light transport in a scene using finite element methods. Goral et al. [GTGB84] described the classical radiosity approach which discretised the scene geometry and computed radiosity for each patch of the surface geometry by solving a system of linear equations. All the surfaces in the scene were considered to be diffuse, hence the radiance and the BRDF were only dependent on the position. Rewriting Equation(2.6) as:

$$L(x) = L_e(x) + \rho(x) \int_A L(y)V(x,y)G(x,y)dA_y$$

where $\rho(x)$ represents the BRDF. Also, $B(x) = \pi L(x)$ and $B_e(x) = \pi L_e(x)$ for diffuse environments. Therefore:

$$B(x) = B_e(x) + \frac{\rho(x)}{\pi} \int_A B(y)V(x,y)G(x,y)dA_y$$

This radiosity equation can be solved by a summation over the patches, using a system of linear equations to compute radiosity, B_i, for each patch of the discretised geometry as:

$$B_i = B_{ei} + \rho_i \sum_j F_{ij} B_j$$

where F_{ij} refers to the form factor between two patches which defines the fraction of power arriving from one patch to the other. The classical radiosity approach first discretises the geometry, calculates the form factors and then solves the system of linear equations using Jacobi or Gauss-Seidel iterations. The two major issues with this approach lie in geometry discretisation and form factor calculation and storage [DBB06]. Research has been carried out to tackle these issues and a review of other approaches can be found in [DBB06, CW93].

The major advantage of computing light transport using radiosity is that it is view-independent and once it has been computed it can be used to generate interactive walk-throughs. Also, form factor calculations can be reused in case of static scenes with dynamic lighting. However, the major disadvantage of radiosity is that it is only applicable for scenes with diffuse surfaces. Algorithms have been developed to incorporate non-diffuse surfaces, for example [ICG86, SP89, SAWG91, AH93], but they usually try to estimate both rendering and radiosity equations, making them computationally expensive.

2.5 Ray Tracing

Ray tracing is a popular technique for generating high-fidelity images which computes the inter-reflections of light around a scene to simulate the light transport. It is an image-order rendering approach [SM09], whereby objects in a scene that are needed for shading computations of a pixel are identified for each pixel of the image to determine its colour value. A light ray is usually traced in a direction opposite to which the light travels, that is from the viewpoint to the light source and hence it is also termed a camera or an eye ray. This is done to prevent wasteful computation by simulating large amount of light rays, all of which might not

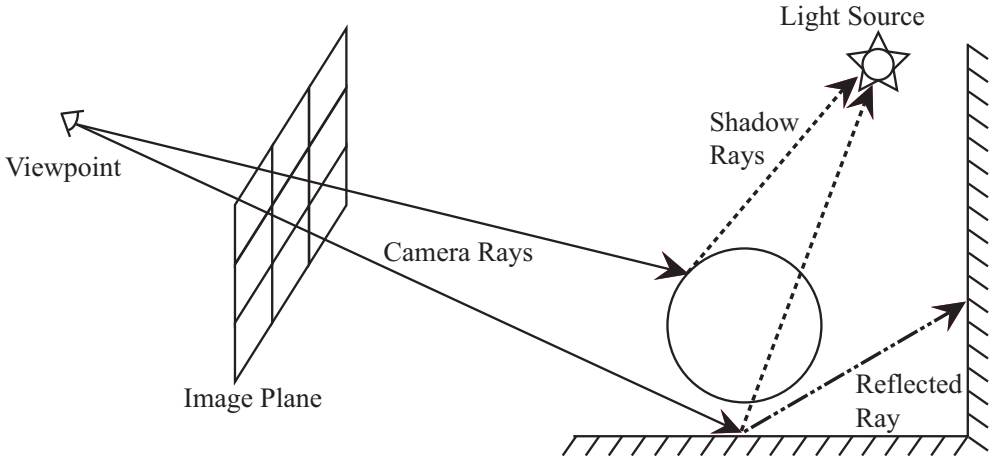


Figure 2.3: Classic ray tracing algorithm

terminate at the viewpoint.

The first ray tracing approach in computer graphics is attributed to Appel [App68] who shot one ray for each pixel to determine the closest intersecting object in the scene for visible surface detection. This approach is known as ray casting and is the simplest form of ray tracing, as it does not propagate the light rays after the first intersection. When light falls on a surface it may be absorbed, transmitted or reflected depending on the surface properties of the material. The classic ray tracing approach [Whi80] on the other hand, simulates the interaction of light with object surfaces by shooting reflected or transmitted rays, recursively upon intersection with a specular surface. Also, at each diffuse intersection point, a shadow ray is shot towards the light source to determine if a point is in shadow. If the shadow ray intersects another object before hitting the light source then the point is considered in the shadow (see Figure 2.3). The classic ray tracing approach is also termed Whitted-style ray tracing.

A basic ray tracer consists of three components [SM09]: ray generation, ray intersection and shading. The first component generates the rays at various steps of the process. The second determines the closest object which intersects with the generated ray. Finally, the last component calculates the pixel colour using information from the ray intersections.

2.6 Point Sampling Methods

Many algorithms with different sampling strategies have been proposed to generate high-fidelity images using point sampling methods, for example: distributed ray tracing [CPC84], path tracing [Kaj86], bi-directional path tracing [LW93], Metropolis light transport [VG97], photon mapping [Jen96], irradiance caching [WRC88], instant radiosity [Kel97], instant global illumination [WKB*02] etc. This section provides an overview of the rendering algorithms which are important from the perspective of this thesis. An overview of the other algorithms can be found in [DBB06].

2.6.1 Distributed Ray Tracing

The classical ray tracing approach generates unnatural images due to the fact that the algorithm supports only point light sources and ideal specular reflection or transmission. This gives rise to artefacts such as hard shadows and mirror-like reflections and does not account for realistic phenomena such as soft shadows, glossy reflections, depth of field, motion blur etc. This limitation in the generation of ray direction was overcome by Cook et al. [CPC84], who accounted for all these subtleties in the light interactions by stochastic sampling, which generated high-fidelity images. Their algorithm is also known as distributed ray tracing.

The distributed ray tracing algorithm employs the Monte Carlo integration technique to estimate the rendering equation. Monte Carlo methods estimate an integral by randomly sampling the function repeatedly and then averaging the results. The larger the number of samples used, the more accurate the estimation. Mathematically, if:

$$I = \int_D f(x)dx$$

then I can be estimated by drawing out N random samples of x from the domain D using Monte Carlo integration as:

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

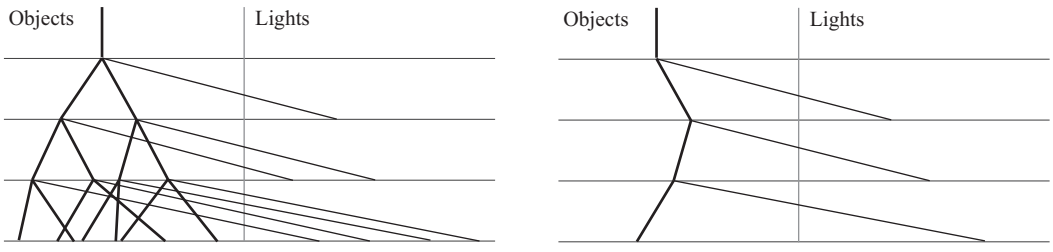
where $\langle I \rangle$ is termed the Monte Carlo estimator of I and $p(x)$ is the probability distribution function for selecting random samples in the given domain. Ap-

plying Monte Carlo integration principle to estimate the rendering equation; the reflected radiance, $L_r(x \rightarrow \Theta)$ (from Equation(2.4))can be estimated as [DBB06]:

$$\langle L_r(x \rightarrow \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x, \Theta \leftrightarrow \Psi_i) L(x \leftarrow \Psi_i) \cos(N_x, \Psi_i)}{p(\Psi_i)}$$

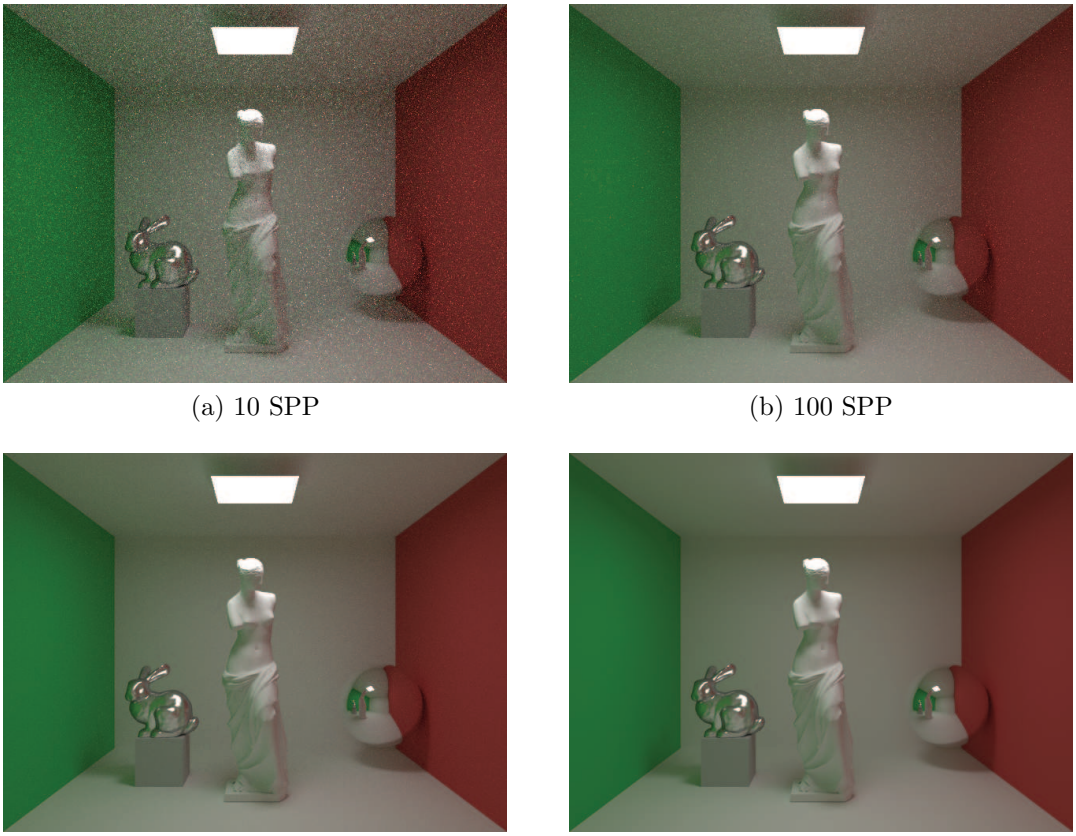
by selecting N random directions, Ψ_i, with probability $p(\Psi_i)$ over the hemisphere Ω_x. However, the incoming radiance $L(x \leftarrow \Psi_i)$ needs to be evaluated. For this, a ray is traced in direction $-\Psi_i$ to estimate the incoming radiance, also known as indirect illumination if the ray does not hit a light source directly. This process is repeated recursively and results in a ray explosion. To curtail the size of the tree, termination conditions such as maximum depth or Russian roulette are employed [DBB06]. The shadow rays are generated at each hit point, similar to the classical ray tracing approach. However, the direction of shadow rays is chosen by sampling different points on the light surface to estimate what is termed as the direct illumination. The process of averaging the radiance using multiple shadow rays accounts for soft shadows.

The variance in the incoming radiance from the different randomly sampled directions gives rise to noise in the resultant image. This high frequency noise is minimised by increasing the number of samples which translates into shooting more rays into the scene at each point. This makes the process computationally expensive for generating high-fidelity images using Monte Carlo techniques and researchers in the field have focussed their efforts at reducing the noise and accelerating the process.



(a) Distributed Ray Tracing (b) Path Tracing

Figure 2.4: Ray traversal based on [Kaj86]



(a) 10 SPP (b) 100 SPP

(c) 1,000 SPP (d) 10,000 SPP

Figure 2.5: Path traced image with different number of samples per pixel (SPP)

2.6.2 Path Tracing

Path tracing is one of the simplest high-fidelity rendering algorithms, introduced by Kajiya [Kaj86]. The distributed ray tracing approach can be considered as an inorder traversal of the tree of rays generated in a scene, while path tracing repeatedly traverses a random path from the root to a leaf node, see Figure 2.4. At each hit point, rather than sampling N random directions as in distributed path tracing, only one direction is sampled for path tracing which prevents a ray explosion and reduces memory requirements. Each path gives a crude approximation of the incoming radiance but averaging many such paths gives a good estimate of the radiance value for each pixel. Furthermore, every path is a Markov Chain through the scene which can be terminated based on criteria similar to those used for limiting the tree depth in distributed ray tracing. A pixel represents a small area of the image and by shooting paths through different points in the pixel area anti-aliasing is obtained. The major disadvantage of using native path tracing

is that it does not employ any optimisations for calculating the light transport and hence it is computationally expensive for obtaining noise-free images. As an example, Figure 2.5 shows the convergence of a path traced image with varying samples per pixel. This algorithm is widely used for generating reference images to compare with other algorithms due to its simplicity and correctness.

2.6.3 Irradiance Caching

Traditionally, in a distributed ray tracing system, a large number of rays are shot at each ray intersection point to sample incoming radiance from the hemisphere around that point. Ward et al. [WRC88], observed that the indirect diffuse component in a scene is a continuous function in space over the surface of an object and unlike specular or highly glossy components, it is not subject to high frequency variations. To exploit this nature of the rendering computation, they proposed an acceleration data structure to store the world space irradiance values computed in the scene. Whenever a new irradiance sample is needed, the cache is consulted to check if there is another sample already calculated in the cache within the user-specified error metric. If there exists such a sample in the cache, the sample to be calculated is interpolated from it, preventing generation of additional rays which would otherwise be traced. The authors show that this strategy of reusing computations provides an order of magnitude speed-up over traditional methods.

The irradiance cache samples are stored in an octree data structure which accelerates the search process for selecting relevant samples. Irradiance gradients [WH92] are employed to choose cached samples which provide a good estimate of the irradiance. Each sample is weighed according to the given formula:

$$w_i(P) = \frac{1}{\frac{||P-P_i||}{R_i} + \sqrt{1 - N_P \cdot N_{P_i}}}$$

where $w_i(P)$ represents the weight of the cache sample P_i for estimating irradiance at point P. R_i refers to the mean harmonic distance of the visible objects from P_i and N_P denotes the surface normal at P. It can be observed that the larger the distance to the cached sample, the lower the weight assigned to it. Also, smaller weight is used for samples whose surface orientation differs significantly. The resultant irradiance at point P is calculated using a weighted average of all the samples in the irradiance cache specified by the user-defined parameter.

The advantages of using irradiance caching are that it is independent of the geometry and as opposed to other methods such as photon mapping and radiosity it is view-driven, conforming to the more traditional ray tracing approach of computing from the point of view of the camera. With a view-driven approach, a walk-through animation path may visit only certain parts of the model being rendered and since this is known beforehand, only those values which are required are computed rather than computing global illumination for the whole model. Moreover, irradiance caching is an established method used in film production [TL04, Her04] and there are many other algorithms based on irradiance caching [TL04, AFO05], adaptations used for dynamic scenes [TMD*04, SKDM05] and adaptive versions of it [YPG01, KBPv06, Deb06, DCG*07]. Yet others perform similar approaches to compute rendering features, such as participating media [JDZJ07], glossy interreflections [KGPB05] and subsurface scattering [KLC06].

2.6.4 Instant Radiosity

Keller [Kel97] proposed a two-pass algorithm for calculating the light transport in a scene. In the first pass of the algorithm, photons are traced from the light source similar to photon mapping or bidirectional path tracing. They are deposited on non-specular surfaces of the scene and act as light sources, termed Virtual Point Lights (VPLs), in the second pass of the algorithm. A GPU is then employed to generate images for each VPL with shadows and the resultant images are averaged to obtain the final image.

Wald et al. [WKB*02] extended this idea for computing global illumination interactively using parallel machines. In the second pass instead of using rasterisation, ray tracing was employed and the VPLs were sampled with shadow rays for estimating the indirect illumination. They used interleaved sampling and filtering techniques to reduce the computational costs. The complete set of VPLs was divided into groups of VPLs and each pixel in a tile used a different group of VPLs to approximate the lighting. The resultant image was filtered using a discontinuity buffer to generate the final image. The advantage of using VPLs for approximating the indirect lighting is that the high frequency noise observed in path tracing type methods is eliminated, resulting in smoother images in less time. However, the reduction in noise is at the cost of artefacts, which can be reduced by shooting more VPLs at the expense of added computation.

2.7 Sparse Sampling and Image Reconstruction

The generation of high-fidelity images using the algorithms described before is a computationally expensive process and research has focused on reducing this cost. An important property of such imagery is its spatial and temporal coherence, that is pixels of an image which are close in space and time are highly correlated. This characteristic of an image can be exploited to reduce the computation by sparsely sampling the image and then using image reconstruction techniques to interpolate the missing pixels. This idea has been widely used for generating high-fidelity images in reasonable times. Quasi-random sequences (see Chapter 4) have been typically used for approximating the solution to the rendering equation by sparse sampling, and Shirley et al. [SEB08] provide an overview of such techniques. However, the primary purpose of employing these sequences in this thesis is to enable fault-tolerant rendering.

2.7.1 Image Reconstruction Techniques

Image reconstruction refers to the process of estimating missing pixels from a set of given pixels in order to complete the image. There are many techniques for reconstructing image pixels and they can be classified into global and local methods. Global methods utilise all the samples present in an image for interpolating the missing pixels.Estimation using local methods is influenced only by samples near the missing pixel. Global methods can be only applied to smaller data sets due to the amount of computational complexity. On the other hand, local methods can be used for reconstruction of larger data sets.

One of the widely used local methods is the nearest neighbour algorithm [Ban09]. In the simplest form, the value of a valid sample, closest to the pixel being estimated, is copied to reconstruct the image. Other variations interpolate the value using k-nearest neighbours, that is they find k closest valid pixels and estimate using different interpolation functions. Figure 2.6 shows an example of k-nearest neighbour algorithm; the black pixels depict valid samples, the grey pixels denote missing samples and the red pixel is the one being reconstructed. The valid pixels inside the dotted circle are used for reconstructing the pixel value.

A variety of functions have been proposed and used for interpolating from k samples. The most straightforward is to average them. Shepard's method [She68]

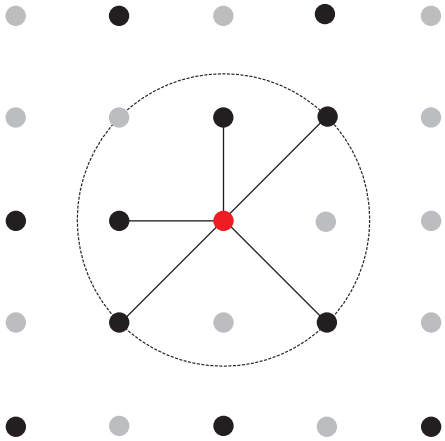


Figure 2.6: k-Nearest neighbour algorithm

uses weighted average, such that farther pixels are assigned lower weights based on the idea that the influence of a sample decreases with distance. The reconstructed value, $s(p)$ of pixel, p using k valid pixels, p_i with values, f_i is given by:

$$s(p) = \sum_{i=1}^{k} w_i(p_i) * f_i$$

where weight, $w_i(p_i)$ is defined as:

$$w_i(p_i) = \frac{h_i(p_i)}{\sum_{i=1}^{k} h_i(p_i)}$$

and $h_i(p_i)$ varies with inverse Euclidean distance, $d(p_i)$ between p_i and p:

$$h_i(p_i) = \frac{1}{[d(p_i)]^{\alpha}}$$

such that $\alpha > 1$. Other functions, such as Radial Basis Function [MDH07], polynomial spline [Doo76] etc. have also been employed for interpolation of two-dimensional data sets. They may offer better interpolation, but they are usually more computationally expensive requiring at least a few seconds to compute [MDH07].

While reconstructing using k-nearest neighbours, care must be taken not to interpolate across edges as this leads to generation of visual artefacts. Bilateral filtering [TM98] is commonly employed to preserve edges while filtering an image. It identifies edges by using a range filter based on the colour values and combines

it with a spatial filter, which for image reconstruction can be k-nearest neighbour. Recently a class of algorithms has been developed, [CPD07, AGDL09, ABD10] which accelerate bilateral filtering by using a multi-dimensional representation of the image. Colour values are also represented as distinct dimensions, in addition to the spatial dimensions. This class of algorithms uses a splat-blur-slice approach to accelerate bilateral filtering, where the multi-dimensional data is projected to a low-resolution spatial data structure, then a Gaussian blur is performed and finally the data is sampled at the original positions.

2.7.2 Sparse Sampling

The generation of interactive visuals has traditionally followed a double-buffering approach, that is two buffers are used for displaying interactive renderings. The front buffer contains the image which is displayed on the user screen and the next image to be displayed is rendered in a back buffer. Once rendering finishes, the back and the front buffers are swapped. This model restricts the interactivity of the system as the minimum time between frames is the amount it takes to render a full frame. For high-fidelity rendering, this imposes a severe constraint and hence the idea of sparse sampling has been successfully employed to overcome this issue. This was first proposed by Bishop et al. [BFMZ94] in the form of Frameless Rendering. Their idea was to compute a fraction of the frame using a highly simplified version of Cook's distributed ray tracer [CPC84]. They rendered a randomised set of pixels, between screen updates using the latest parameters for rendering, to prevent image tearing, and a simultaneous partial update of the full image. In the transient state, this appeared as motion blur and when a stable state was achieved, the image was eventually fully rendered.

The idea of Frameless Rendering was further improved by the Render Cache algorithm [WDP99]. It stores every sample generated from the ray tracing engine into a data structure containing information about the hit point, colour and its age. These samples are reprojected onto the current view and depth information is used for occlusion. Any holes in the resulting image are filled by using linear interpolation and filtering. Also, new samples are recalculated after they age beyond a particular threshold. Heuristics are employed to prematurely age samples which are in the areas most likely to change, hence offering a progressive refinement of the view. The rendering process is decoupled from the viewing process and they are performed asynchronously. However, the system is unable to deal

with quick changes and the transient images show visible artefacts. Walter et al. proposed an improved version of the Render Cache algorithm [WDG02] by adding an additional filtering pass and lowering the visual artefacts by employing predictive sampling. Furthermore, they optimised the algorithm for coherent memory access to gain speedup. Other similar techniques have been presented by Ward and Simmons [WS99], and Simmons and Séquin [SS00].

The Render Cache methods used image-based techniques for interpolating from sparse samples. Tole et al. [TPWG02] improved on this idea by performing reconstruction in the object space using a Shading Cache. Their data structure stored the samples on the scene geometry and used GPU for interpolating missing samples. This helped in limiting the visual artefacts, however it took time to update global illumination effects in dynamic scenes with moving lights and objects. Bala et al. [BWG03] extended the Shading Cache by preserving discontinuities during the image reconstruction process. They determined object silhouettes and shadow edges in real-time and prevented reconstruction across these boundaries for a better visual fidelity. These algorithms have been implemented and enhanced using the advanced flexibility provided by modern day GPUs and presented in [VALBW06, SaLY*08].

Dayal et al. [DWWL05] presented ideas to optimise the process of generation of new samples and the reuse of older samples in the form of Adaptive Frameless Rendering. They used a guidance mechanism to adaptively generate new samples based on spatio-temporal colour variations. Furthermore, they employed filtering in temporal domain based on temporal gradients and implemented the reconstruction process on GPU. All the methods described above provide a substantial gain while presenting visual feedback to a user, however they suffer from distracting artefacts such as image tearing, missing pixels etc.

Adaptive or progressive rendering techniques, for example [BFGS86, Mit87, PS89, LWC*02], are also similar to sparse sampling methods. These compute a coarse approximation of the rendered image and then refine it over time. While using such algorithms in a double-buffering approach, it is very important to decide when to switch between the buffers. Woolley et al. [WLW02] presented a scheme for making this decision, which they termed interruptible rendering. The idea was to continue the rendering process until the temporal error exceeds the spatial error. The temporal error refers to the error which is the result of time it takes for the computation of an update from when the user triggered it and the spatial error refers to the error due to the coarseness of the rendered image.

2.8 Time-constrained Rendering

Rendering systems have often been subjected to time-constraints due to the various possibilities of continually adapting and refining the computation. The use of time-constraints thus allows algorithms to be designed such that they aim to achieve the best possible quality within a given time period. These become extremely useful in the case of interactive rendering, where screen updates are generally required at a fixed rate to maintain smooth interactivity.

Funkhouser and Séquin [FS93] devised a mechanism for rendering models interactively using rasterisation at a fixed frame rate. The models in the scene were pre-processed at discrete level of detail and their greedy algorithm predicted which version to render in order to maintain the time-constraint. This was done to maximise the benefit obtained by rendering at a particular level of detail, such that the cost associated with rendering at that refinement was less than the constraint. Maciel and Shirley [MS95], and Mason and Blake [MB97] further extended this idea by creating a single hierarchy of all the objects rather than treating each object individually and used variations of multiple choice knapsack problem for selecting the appropriate representation. Gobbetti and Bouvier [GB99] enhanced this approach by using continuous level of detail models. Zach et al. [ZMK02] presented an algorithm which incorporated both continuous and discrete level of detail for rendering of terrains at a guaranteed frame rate using polygonal and point-based rendering. An importance metric was presented by Gao et al. [GLH*08] for distributed visualisation of large data sets to determine the rendering order and the level of detail for a block of data set based on view-dependent, application-dependent and data-dependent criteria.

Reisman et al. [RGS00] used time-constraints for interactive parallel ray tracing. They used a progressive sampling strategy based on Delaunay triangulation while treating the image plane as a continuous space. They refined their solution until a given deadline and then reconstructed the image from calculated samples using piecewise linear interpolation. Debattista et al. [DSSC05] provided a framework for controlling the pixel quality in a time-constrained setting for generation of high-fidelity images without perceivable difference. They proposed a regular expression to specify the pixel computations based on different components. Debattista [Deb06] further enhanced the approach by using time-constraints with a progressive selective rendering pipeline.

2.9 Parallel Rendering

The large computational complexity exhibited by high-fidelity rendering has encouraged researchers to tackle the rendering problem with parallel computing. An early survey of parallel rendering techniques and related issues is provided by Crockett [Cro97]. A detailed survey of these methods especially in context of global illumination and ray tracing is presented by Chalmers et al. [CDR02]. Ray tracing algorithms are relatively easy to parallelise if the entire scene description can be duplicated on each processor, as each processor can be designated to independently work on a part of the image-space. However, load balancing can be challenging.

Parallel rendering tasks can be constructed by subdividing either in image-space or object-space. The former is suitable if the scene data can be replicated on each processor and the latter is generally used when the scene data is larger than the memory size of an individual processor. A uniform load balancing is easier to achieve with image-space subdivision (for example [Woo84, PB85, LS91]) but it leads to poor memory usage. In contrast, a scene can be distributed evenly using object-space subdivision (for example [DS84, PB89, Pit93]) but it results in complex load balancing strategies. Many algorithms have been proposed in both categories taking advantage of hardware and application environments. A hybrid task subdivision strategy was developed independently by Salmon and Goldsmith [SG88], and Scherson and Caspary [SC88], where object data is hierarchically subdivided. The upper portion of this hierarchy is replicated on all processors while the lower part with most of the scene data is distributed allowing dissociation of image-space and object-space.

Load balancing can be static or dynamic based on the scheduling policy adopted by the algorithm. Heirich and Arvo [HA98] provided a good comparison of various load balancing techniques and showed that any static task subdivision strategy is affected by load imbalances. In their experiments, they observed that a hybrid method reduced the load imbalance cost such that it was smaller than other costs of parallelisation. Badouel and Priol [BP89] described a dynamic demand-driven load balancing based on the master-worker paradigm whereby each worker is assigned a 3×3 tile of pixels when it becomes idle. But this approach suffered from poor scalability and therefore, Green and Paddon [GP90] used a hierarchical approach to overcome this. They connected the distributed processors in a tree structure such that the master process was at the root of

the tree and all the other workers would communicate with their parents only, limiting the number of requests being sent across the network. This resulted in better scalability. Reisman et al. [RGS00] devised a dynamic load balancing strategy for progressive ray tracing on distributed clusters exploiting temporal coherence for obtaining interactive rates. The image was subdivided into regions which were assigned to separate processors and during run-time the regions were dynamically adjusted to rectify load imbalances.

One of the first parallel architectures designed for interactive ray tracing using a 96-processor supercomputer was proposed by Muuss [Muu95]. He needed interactive rendering for radar systems which had been modelled using large number of Constructive Solid Geometry (CSG) primitives. These could not be rasterised as the tessellation of the primitives would have resulted in several million polygons. Using traditional ray tracing the author was able to render the CSG primitives directly, preventing the memory and time costs of tessellation. Keates and Hubbold [KH95] presented an interactive ray tracing system based on a virtual shared-memory machine which took advantage of cache coherence by using regular gridcells. Dynamic load balancing was achieved with thread synchronisation mechanisms to achieve interactivity on 64 processors.

Another such rendering system was proposed by Parker et al. [PMS*99] using an optimised static load balancing approach. This supported image-based rendering with realistic shadows which also ran on a shared-memory supercomputer. Interactivity was achieved by designing the jobs to match the caching granularity and exploiting fast synchronisation available on the target hardware. They were also able to perform volume rendering [PPL*99] and iso-surface visualisation [PSL*98] using their system which supported frameless rendering in addition to synchronous operation.

Wald et al. [WSBW01] enhanced the methods presented in [PMS*99] by using object dataflow strategy to run on a distributed cluster. Using a highly optimised SIMD code they were able to achieve interactivity. Wald et al. [WKB*02,BWS03] further enhanced this by adding global illumination and handling a few dynamic scene changes by modifying instant radiosity (see Section 2.6.4).

Purcell et al. [PBMH02] exploited the programmability of the modern GPU and used it as an efficient and massively parallel stream processor for ray tracing, demonstrating similar performance to the optimised CPU version of Wald et al. [WKB*02]. They moulded the ray tracing algorithm into a streaming process by breaking it down into a set of kernels representing different aspects of the

computation.

Yao et al. [YPZ10] presented a system for parallel animation rendering on distributed resources. In addition to rendering frames in parallel, their system used tile-based image subdivision for finer granularity to achieve dynamic load balancing. Their system was designed to take advantage of the spatial and temporal coherence between animation frames while scheduling parallel tasks.

2.9.1 Parallel Irradiance Cache

There have been several attempts at parallelising the irradiance cache. The standard Radiance light simulation package [LS98], supports a parallel renderer which uses the Network File System (NFS) to manage simultaneous access to irradiance cache between parallel processes on a distributed system. This may lead to file contention while using inefficient file lock managers resulting in poor performance. Koholka et al. [KMG99] shared the irradiance values between processes on the slaves using MPI after every 50 irradiance samples were calculated at each slave. Robertson et al. [RCLL99] proposed a master slave model of Radiance where each slave calculates and deposits the irradiance cache values to the central master after a threshold, and then gathers the irradiance values calculated by other slaves from the master after a threshold. Debattista et al. [DSC06] followed the irradiance caching component-based philosophy by subdividing the computation of the indirect diffuse on a separate dedicated set of irradiance cache nodes, while the remainder of the nodes computed the rest of the rendering. This enabled cached samples to be shared quickly and efficiently amongst the dedicated irradiance cache nodes.

2.10 Summary

This chapter has described the theoretical concepts and methodology required for high-fidelity rendering. Point sampling methods for solving the rendering equation, which have been adapted for shared computing resources as described in this thesis, have been presented. Previous techniques for accelerating rendering by employing sparse sampling, image reconstruction and parallel rendering have also been discussed. The next chapter will provide an overview on shared computing resources and also explain why the traditional rendering algorithms are not suited for using such parallel resources.

CHAPTER 3

Computing on Shared Resources

This chapter presents an overview of shared computational resources. It begins by categorising different types of grids before describing computational and desktop grids. Subsequently, traditional fault-tolerance techniques: replication and checkpointing are discussed. Next, the master-worker paradigm for computing on shared resources is presented. This is followed by a description of two grid middlewares: Globus and Condor, which have been employed in this thesis. Finally, a discussion on parallel rendering on shared resources is presented.

3.1 Taxonomy of Grid Computing

Grid computing can be classified into two broad categories: computational grids and desktop grids (see Figure 3.1). A computational grid usually refers to sharing of dedicated resources while a desktop grid refers to sharing of non-dedicated resources. A desktop grid can be further distinguished into Internet-based and LAN-based, depending on the type of interconnect used for communication be-

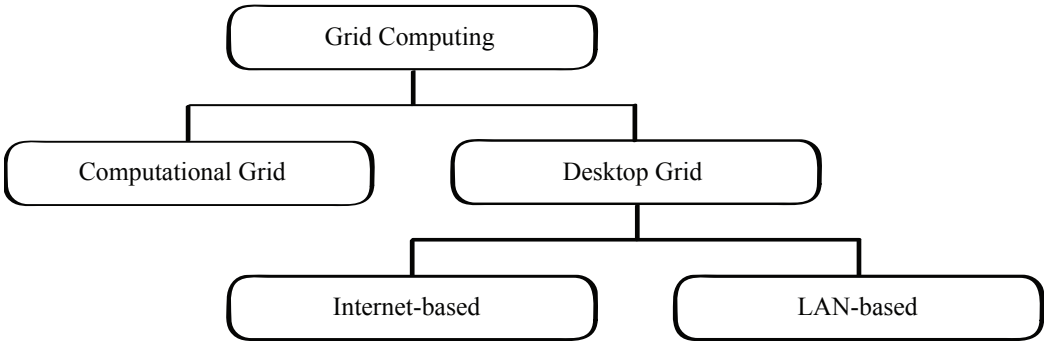


Figure 3.1: Grid computing taxonomy

tween the resources. The following subsections present a further comparison among these types of grid computing to elucidate the differences.

3.1.1 Computational Grid vs. Desktop Grids

The main point of differentiation between a desktop grid and a computational grid is the underlying resources which provide the computational power. A desktop grid is powered by idle CPU cycles of commodity workstations while a computational grid usually uses compute cycles from clusters and supercomputers. A computational grid may additionally also combine databases and scientific equipment, which are shared across institutions. These resources are connected via dedicated high speed networks. In contrast, desktop grid resources are connected via non-dedicated networks.

The volatility of desktop grid resources is considerably higher than those on a computational grid. This is due to the fact that the resource providers in a computational grid are trustworthy in comparison to a desktop grid and they are sometimes required to commit to service level agreements before being allowed to be a part of it. On the other hand, the resources in a desktop grid only provide compute power when not being used by the owner, which is the root cause for volatility of such resources. Furthermore, computational grids sometimes provide facilities for reserving the resources in advance such that they can be dedicated for a specific user application. Hence, computational grids are more reliable than a desktop grid. This attribute is further corroborated by the fact that computational grids have been targeted towards high performance computing while desktop grids have been typically employed for high throughput computing [Cho07].

3.1.2 Internet-based vs. LAN-based Desktop Grids

A desktop grid can be further distinguished into Internet-based and LAN-based [Cho07]. An Internet-based desktop grid is powered by CPU cycles donated by desktop machines of volunteers across the globe, while a LAN-based desktop grid typically uses workstations within an institution such as a university or an office. An Internet-based desktop grid is the cheapest option for grid computing from the user perspective, as the resources are owned and maintained by volunteers. But the main issue is that the user application needs to be appealing enough to gen-

erate sustainable interest from volunteers, for example SETI@Home [ACK*02], FightAIDS@home [CLOB07] and Genome@home [LSS*04]. The primary function for machines on a LAN-based desktop grid is not to be a part of it, but they may be personal workstations of the staff in an institute or machines in a student lab or an office. Hence, they are suitable for running any applications which the institution deems useful.

An application employing an Internet-based desktop grid generates interest by giving credits to the volunteers for successfully completed computations. The application thus needs to incorporate result verification strategies to prevent against malicious volunteers who tamper with the normal execution of the application to gain more credits. This results in a loss of useful computation, since many copies of same task are generally computed and the most common result is accepted to be correct [DSMS07]. In contrast, as a LAN-based desktop grid is created by an institutional policy rather than on voluntary basis, and thus such a need for cross checking the results does not arise. Furthermore, Internet-based desktop grid applications need to support a large variety of platforms and processors while the heterogeneity of resources in a LAN-based desktop grid is limited. As an example, BOINC projects [And04] list about 4400 variations of host CPUs and 64 different operating systems [BOI10].

3.2 Computational Grid

A computational grid can be defined as a distributively-owned multi-programmed large pool of heterogeneous computing resources interconnected by telecommunication channels. Theoretically, it covers a wide assortment of concepts for computing on shared resources. The motivation for computational grids has been driven by the need for computational power in a fashion similar to an electrical power grid, such that the user is unaware of its origin and has an easy access to it. Foster and Kesselman [FK99] envisaged a computational grid as:

"... a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities"

A computational grid needs to be dependable, so that the users are guaranteed access to resources for their computational requirements whenever the need arises. Consistency refers to the necessity that standardised services are provided to the user for a widespread use. Finally, large computational power at a low cost would

make them pervasive.

In a further refinement, Foster and Kesselman [FK04] defined a computational grid as:

"... a system that coordinates distributed resources using standard, open general-purpose protocols and interfaces to deliver nontrivial qualities of service."

A grid uses security protocols for user authorisation and authentication while protecting its resources from malicious users. There are protocols for resource and task management. Their emphasis was on standardisation of these protocols for permitting dynamic resource sharing between all those who are interested rather than producing isolated, discrepant and non-interoperable distributed systems. Any entity which would become a part of the computational grid would support such standard protocols, enabling effective sharing.

An organisation which defines the rules and conditions for sharing resources between a dynamic group of collaborators in a computational grid is termed as a Virtual Organisation (VO) [FKT01]. The entities of a VO have common interests for which they pool their resources, knowledge and competencies, and cooperate to help each other. All VOs can differ significantly in number and nature of participants, the level of sharing, the duration of time for which they exist and the organisational structure, but it is still possible to identify similarities between them. They all need flexible sharing mechanisms which can protect local autonomy while sharing diverse range of resources using well-defined control policies. A computational grid serving a variety of users can consist of multiple VOs, for example biomedical scientists and geologists can form different VOs but can still use the same grid infrastructure for collaboration. Also, a single member can be affiliated to multiple VOs and the computational grids are expected to seamlessly provide access to the required resources from different VOs based on the needs of the user.

Foster and Kesselman [FK04] proposed a four-layered hourglass model (see Figure 3.2) of computational grids to identify its components and their interactions. The fabric layer encompasses the resources such as processors, databases, networks and instruments at the bottom of their architecture. The top most layer i.e. the user applications, harnesses these resources using the two layers in the middle: collective services for combining the resources and the resource and connectivity protocols. The idea is to have a limited set of protocols and services which can be used for aggregating large number of resources for executing wide variety of user applications.

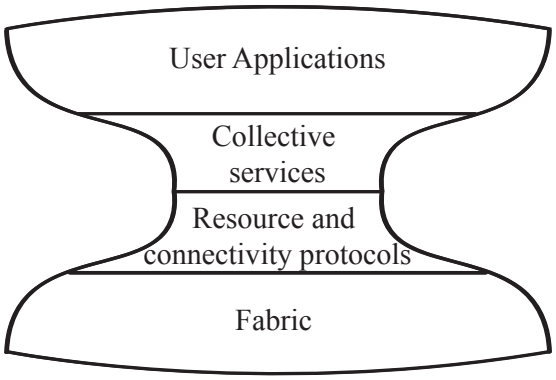


Figure 3.2: Four-layered hourglass grid architecture based after [FK04]

3.2.1 Examples

The National Grid Service (NGS) [NGS10] is a computational grid formed by collaboration of various academic institutions across the United Kingdom. Its computational infrastructure, which consists of both clusters and desktop machines, is maintained by the participating institutions. The NGS provides computational and data storage facilities, free of cost, to any UK academic for carrying out their research. A variety of application tools in areas as diverse as astrophysics, biomedical, engineering, image analysis, chemistry, bioinformatics have been installed on the NGS to help the users. TeraGrid [TER10] and the EGI (European Grid Initiative) [EGI10] are two major grid infrastructures, similar to the NGS, for aggregating resources in the United States and Europe respectively.

The Large Hadron Collider (LHC) is a high-energy particle accelerator operated by the European Organization for Nuclear Research (CERN) at the Swiss-Franco border. The particle detectors installed at the LHC for experiments such as ATLAS, ALICE, CMS and LHCb would generate about 15 Petabytes of data annually [WLC10]. Hence, the massive amount of computational power needed to process this data led to the formation of a global collaboration known as the Worldwide LHC Computing Grid (WLCG) [WLC10]. It consists of around 170 participating institutions across 34 countries with more than 100,000 processors. The aim of WLCG is to provide the infrastructure for high-energy physicists around the world to access and analyse the data generated by LHC.

Hybrid experiments in earthquake engineering are carried out by combining a computational simulation model and data from physical experiments to study the behaviour of earthquakes. Some of these physical experiments require apparatus owned by different institutions, unavailable at a single geographical location. The

Network for Earthquake Engineering Simulation Grid (NEESgrid) [PKG*04] was formed to overcome this challenge of coupling such experimental apparatus and computational resources for conducting the experiments. It is based at the San Diego Supercomputer Center, United States and provides tools for earthquake engineers to run hybrid simulations and collaborate with fellow earthquake engineers.

3.3 Desktop Grids

Desktop grids are based on the idea of harnessing idle compute cycles from the CPUs. This idea of cycle stealing from desktop machines originated with the PARC worm [SH82] which scanned a list of resource addresses and replicated itself on idle hosts for testing distributed applications. Since then, this idea has been widely used for solving complex scientific problems such as SETI@Home project [ACK*02], Folding@Home [LSS*04], GIMPS [GIM10], FightAids@Home [CLOB07]. The main purpose of the resources on a desktop grid is not to provide compute cycles for running desktop grid applications, only their spare compute cycles are used.

With the advent of multi-core desktop machines, desktop grids are becoming an attractive option for distributed computing. The routine activities performed on a workstation such as word processing, web browsing, emails, instant messaging, documentation etc. typically have minimal CPU requirements. In fact, usually activities which rely on human intervention hardly stress the modern machines. Desktop machines in an institution connected to a LAN such as those in student computer labs and offices are ideal for use as part of a desktop grid. It has been observed that such machines exhibit an estimated CPU idleness of up to 95% [DMS05, Hea03]. Kondo et al. [KFC*07] approximated that a 220 hosts desktop grid was equivalent to a 160 hosts cluster on weekdays and a 209 hosts cluster on weekends.

While most of the users do not utilise the available CPU power, some researches are limited by the amount of computational power available to them. A big category of problems in areas as diverse as cryptography, earth sciences, cosmology, high energy physics, biology and medicine need large amount of computational power to be tackled efficaciously [Boh05]. Therefore, users who do not have access to high performance machines can benefit by employing idle CPU cy-

cles. Furthermore, even though applications need to be modified to run on desktop grids, the return on investment for such applications is usually high [And04].

Kondo describes the following characteristics for an ideal desktop grid [Kon05]: scalability, fault-tolerance, security, manageability, unobtrusiveness and usability. To elaborate, the throughput of a desktop grid should scale proportionally to the increase in the number of resources. A fault-tolerance strategy is needed such that the desktop grid does not fail in case any of the resources fails. The application and the donor's resources must be mutually protected. Tools should be provided for installing, updating and monitoring the compute resources as human resources are more expensive compared to them. An important characteristic specifically for desktop grids is the unobtrusiveness, that is the donor's tasks have the first priority and the user's application must not interfere with them. Finally, the process of porting user applications to run on a desktop grid needs to be transparent in order to make them usable.

3.3.1 Advantages

The most significant advantage of a desktop grid is their lucrativeness. The users of a desktop grid do not need to invest in the hardware which provides the computational power. On an Internet-based desktop grid, the volunteers bear the cost of maintaining and upgrading the hardware. On a LAN-based desktop grid, the cost is borne by the institution which operates the machines. In fact, building desktop grids from already existing infrastructure provides a mechanism for the institutions to utilise their resources more effectively. Some additional infrastructure may be required for constructing a desktop grid, such as a few servers for installing the middleware and administrators for managing them. But this extra cost helps in better utilisation of resources which further increases the return on investment [And04].

The second advantage of employing desktop grids is the high amount of computational power that can be used for problem solving which otherwise would have been usually wasted due to idleness. The BOINC projects utilise 3.0 PetaFLOPS from about 490,000 active hosts worldwide [BOI10] which compares favourably to the 2.3 PetaFLOPS [TOP10] provided by the fastest current generation supercomputer. This computational power of desktop grids increases over time, without any investment from the user side, as the volunteers upgrade their hardware.

3.3.2 Challenges

Even though there are clear benefits of using desktop grids, there are several challenges which must be addressed. The major challenge that users face is the volatility of resources. Resources which provide the computational power to a desktop grid are non-dedicated. They may be employed only when the owner is not using them. This means that they are donated without any guarantee of service. Choi et al. [CBH*04] defined two classes of failures for desktop grids: volatility failure and interference failure. The first refers to a hardware failure of a resource. As large number of resources are employed in a desktop grid the probability of such failures is high. The second class of failures occur when the owners claim back their resource and the task submitted by the user gets evicted. The grid middleware is usually responsible for ensuring that any foreign task does not interfere with owner's tasks and allows the owners to specify the conditions which define the sharing. For example, Condor [CON10] (see Section 3.6.2) monitors user input and CPU usage for deciding which machine can be considered idle. Many studies (for example [Kon05, DMS05, WSH00, Din99, LMG95, ML91]) have been carried out to understand this volatility of resources in different environments for fine tuning the user applications for maximum performance.

The connectivity of resources poses another challenge while adapting applications for use on desktop grids. As the networks which connect the resources of a desktop grid are not dedicated and often have high latencies and low bandwidth, frequent communication between the parallel processes is not advisable [Mes99]. The applications which can be parallelised using a bag-of-tasks model, that is they can be subdivided into independent loosely-coupled tasks, are well suited for desktop grids [CBS*03]. Firewall rules often allow communication only over well known ports which may restrict the communication between resources and the users. Also, use of Network Address Translation (NAT) prevents identification of hosts by a unique IP address. These restrictions have a more pronounced effect on Internet-based desktop grids compared to LAN-based ones. The network architecture restricts applications which are parallelisable using tightly-coupled tasks from fully harnessing the desktop grids. However, there still exists a large category of problems which can be parallelised by overcoming these challenges in communication to run on them.

A resource may be harmed unintentionally by a faulty application which can disrupt the normal execution of the machine or intentionally when a user task may

try to retrieve information about the donor. On the other hand, a donor might try to intentionally alter the normal execution of a user application to earn extra credits or wrong results may be generated unintentionally due to faulty hardware. Therefore, user applications and the donor's resources need to be mutually protected from each other against intentional and unintentional harm. The applications often use security mechanisms such as encryption, code signing, validation and consensus before accepting any results from the users [Dom08]. Techniques such as sandboxing [CCEB03], resource virtualisation [GN07], to name a few, are usually employed for protecting the resources. These security concerns are higher for an Internet-based desktop grid rather than a LAN-based desktop grid due to a lower level of trust between the users and the donors in the former case.

The heterogeneity of resources presents another challenge while developing applications for desktop grids. This entails that the application code needs to be portable for executing in different environments which exist on a desktop grid. The use of interpreted code can overcome portability problems, but it is affected by poor performance and overheads which are associated with such systems. For example, System Virtual Machines may be employed which allow the users to precisely control the execution environment and protect the resources as well [FDF03]. The drawback of such techniques is that they require specialised compatible software to be installed and maintained on all the resources. The heterogeneity can also affect the accuracy of results generated by the tasks due to various hardware [TACBI05]. Proper load balancing in accordance with the different capabilities of the underlying hardware can have a great effect on overall efficiency.

3.4 Fault-tolerance

Fault-tolerance of a system is its ability to mask the presence of faults in the system while aiming to prevent system failure [Jal94]. A system is considered fault-tolerant if it performs according to its specifications in the presence of internal faults. Fault-tolerance is a well studied topic in the field of distributed computing [CDK01, Jal94, AL81, Avi76] and various mechanisms have been developed for tackling faults. It is an important aspect for the dependability of a system [ALR04], that is the system can be trusted to perform its functionalities, which results in its increased usability. In case of grid computing, the impor-

tance of fault-tolerance is even more significant as faults in grids are a natural occurrence rather than an abnormal phenomenon. The sharing of large number of distributed resources in a grid and their volatility, especially in desktop grids, makes them prone to faults.

Cristian et al. [CAS86] classified faults based on the behaviour of a failed component into four categories: crash, omission, timing and Byzantine. Crash fault occurs when the component's internal state gets corrupted and it stops working. An omission fault causes the component to respond intermittently. When a component does not respond within a specified period, a timing fault occurs. A Byzantine fault refers to any arbitrary fault which leads to non-deterministic behaviour of a system and this category of faults is the hardest to detect. A more comprehensive classification of faults can be found in [ALR04].

There are four stages in any fault-tolerance mechanism [Jal94, AL81]: error detection, damage confinement, error recovery, and fault treatment and continued system service. The first stage deals with identifying any errors in a system, indicating the presence of faults. In the second stage, effects of the error are limited from propagating within the system. Subsequently, in the third stage actions are taken for error removal and restoring the system. Finally, the faulty component of the system is identified and measures are taken such that it is not used again in a manner which might affect the functioning of the system. These stages can be explicit or be implicitly combined in a fault-tolerance mechanism, but the sequence of actions usually remains unaltered.

3.4.1 Fault-tolerant Strategies

Fault-tolerance methods can be broadly classified in two main categories: replication and checkpointing. Replication uses multiple instances to create a backup mechanism which can be used in case one instance fails, while checkpointing stores a copy of the system state periodically from which the system can be recovered when a fault occurs. A few strategies use a hybrid of these two approaches for creating multiple levels of fault-tolerance. The following subsections discuss these strategies from the perspective of preventing failure of user tasks in case of faults in a grid.

3.4.1.1 Replication

In grid computing, replication is used to compute multiple copies of user tasks such that when one copy fails another copy may be used. There are two kinds of replication techniques: active and passive. In an active replication strategy, all copies of the tasks are scheduled together. This is used in cases where the cost of computing replicas of the tasks is small compared to the amount of computational power available. Passive replication uses a single copy of the user task and replicates it only if the primary task fails. This helps in minimising the waste of useful computational power. However, in case of deadline-driven environment, passive replication can be hindering as the backup copies are scheduled only after detection of failure. A hybrid approach, by scheduling primary and backup copies overlapping in time, is used in real-time systems for better fault-tolerance [AOSM04].

Replication of tasks is also used for generating multiple results which help in prevention against malicious intent, especially in the case of Internet-based desktop grids [DSMS07]. Incorrect results can be classified as Byzantine faults and it can be shown that $2f + 1$ replicas are needed to recover in the presence of f Byzantine faults [CDK01]. Kondo [Kon05] described two challenges related to task replication and developed heuristics for tackling these in a desktop grid environment with an aim of reducing the turnaround time. The first challenge is to identify which tasks need to be replicated and on which resources and the second issue is to determine how many replicas need to be created. Generally, researchers have aimed at minimising the impact on performance due to replication by carefully addressing these two issues.

3.4.1.2 Checkpointing

Checkpointing schemes allow resumption of computation from the last state which was successfully saved before the failure occurred. This may reduce the loss of useful computation which happens due to replication but can impose additional overheads for storing the computation state. They can be either stored at the system level or at the application level [SS98]. A system level checkpoint saves the process state of the task, raising portability issues. Although it is advantageous from the perspective that application-specific knowledge is not needed, making it a generalised mechanism. Application level checkpointing saves out application-specific data from which the computation can be resumed in case of

a failure. This needs application-specific knowledge and hence requires a specialised solution for each application, but it is usually portable across various resources.

In a distributed system, the challenge lies in achieving a uniform view of the system's state across all nodes [CL85]. Elnozahy et al. [EAWJ02] presented a survey of protocols for rollback and recovery and specified two types of checkpointing: uncoordinated and coordinated. In uncoordinated checkpointing, each component checkpoints independent of each other while in coordinated checkpointing all the components synchronise to save a globally consistent state. The nature of grids make it difficult to use coordinated checkpointing as synchronisation across all resources has large overheads. Hence, uncoordinated checkpointing by saving individual checkpoints to a universally accessible place is generally preferred.

3.5 The Master-Worker Paradigm

The task execution model of grid computing, especially for desktop grids, often relies on a master-worker paradigm. In this type of work distribution model, a user submits tasks to the master which may subsequently divide them into a set of loosely-coupled independent subtasks (also termed bag-of-tasks model). The workers then request these tasks from the master, process them and send back the results. The master usually executes on a resource which is dedicated and accessible from all the workers. The workers can run on any other resource in the grid which may be non-dedicated. Once all the tasks have been processed by the workers, the master can combine the results.

There are many advantages of using the master-worker paradigm. A fair load balance is achieved, enabling effective sharing of resources, by maintaining a centralised pool of tasks on the master from which the workers can request tasks, when idle. As the workers do not need to communicate with each other, they can be on separate networks. Furthermore, the availability of a worker can change during the course of a computation without hampering its overall progress, thus allowing the donors to provide resources without any guarantee of service. In the case a worker fails to complete a task, replication can be used to assign the task to another worker from the resource pool making the system fault-tolerant. In many organisations, firewalls block incoming requests but allow any outgoing

connections. Therefore, a task pull mechanism allows the workers to initiate the communication with the master and overcomes firewall problems on machines with such an asymmetric network connection.

The master-worker paradigm suffers from a centralised point of failure as the master stores the complete progress of the computation. Fault-tolerance of the system can be enhanced by replication or checkpointing the master so that it remains functional even when the master fails. Also, the scalability of the computation can be affected by having a single master as it may be unable to handle increasing number of workers. If this becomes an issue, a hierarchy of masters can be employed as described in [Ban06].

3.6 Grid Middleware

A grid provides a variety of services to the users which can be broadly classified into four types: task management, data management, security and monitoring. The task manager is responsible for accepting tasks from the users and scheduling them on appropriate resources. The data manager provides tools for transferring data across various locations in a secure and reliable manner. Security services allow verified users to access the resources and delegate their credentials to appropriate resources for performing the tasks on their behalf. The monitoring services inform the users about the status of resources and perform accounting and bookkeeping functions for a fair usage. All these services are essential for sound operation of any grid. The software infrastructure which performs these services is termed grid middleware. The following subsections provide information on two relevant grid middlewares from the perspective of this thesis. A detailed and comprehensive review of other desktop grid middlewares can be found in [Cho07].

3.6.1 Globus Toolkit

One of the fundamental requirements for building an ideal computational grid is to develop standard protocols for grid services. The Globus Alliance [GLO10] is an international association to build these standards and technology. The Globus Toolkit [Fos06] is an open source implementation by them and is considered the de facto standard for computational grids. The toolkit is based on the Open Grid Services Architecture (OGSA) [FKNT02] and the Web Services Resource

Framework (WSRF). It is a software package which allows users to build computational grids by sharing their resources with each other without giving up their local autonomy.

The suite of tools includes components that implement grid services such as resource management, data management, communication, fault detection and security. Developers can choose from these components to build their own grid applications and services without worrying about the underlying protocols. Many commercial vendors such as Hewlett-Packard, IBM, DataSynapse, United Devices are using the Globus Toolkit for creating such applications. It is also being used all over the world by large scale production computational grids such as National Grid Service (NGS) [NGS10], TeraGRID [TER10], Network for Earthquake Engineering and Simulation Grid (NEESgrid) [PKG*04] and Worldwide Large Hadron Collider Grid (WLCG) [WLC10]. The main components of the Globus Toolkit include [Fos06]:

Grid Resource Allocation and Management (GRAM): This service is responsible for task submission, monitoring and cancelation on the local resources. The toolkit does not provide a task scheduling mechanism and instead uses this service to communicate with the local schedulers and supports Portable Batch System (PBS), Platform LSF and Condor. It facilitates the delegation of credentials and file transfers from the submitting host to the execution host.

Grid Security Infrastructure (GSI): The security service of the Globus Toolkit uses asymmetric cryptography for authenticating users and services. A security certificate encoded in X.509 format and signed by a Certificate Authority (CA) is used for secure communication between the entities of the computational grid. It provides a single-sign-on and credential delegation using proxy certificate which are valid for limited time period without having a central security system.

Grid File Transfer Protocol (GridFTP): A large amount of data needs to be transferred between resources in a computational grid which may lie on different networks. GridFTP is a protocol for a fast, secure and efficient transmission of bulk data between the grid elements. It uses advanced file transfer mechanisms such as data stripping across multiple connections for enhanced performance. Along with the Reliable File Transfer (RTF)

service and Replica Location Service (RLS) it forms the complete set of data management services needed to build a computational grid.

Monitoring and Discovery System (MDS): This is a system which provides the status information of resources and locates new resources on a computational grid by identifying resources which form a part of the VO. It is composed of two services: an index service and a trigger service. The index service gathers information by querying the local information providers and the trigger service launches any actions that might be needed based on the received information. The users and services such as resource brokers, can query the MDS to check available resources. It acts as a central data manager for publishing information such that it becomes available to multiple sites on a computational grid.

3.6.2 Condor

Condor [CON10] is an open source high throughput computing platform developed at the department of Computer Science, University of Wisconsin, Madison. It is a middleware for seamlessly tapping unused CPU cycles in an institutional environment and provides a batch system for job queuing, execution, management and monitoring on vacant resources. The resources can be either idle desktop machines or dedicated cluster nodes. The aim of the project is to deliver a large number of computation cycles to the users over a long period of time in contrast to high performance computing which focuses on delivering high computational power over a shorter time period. Condor supports popular operating systems (Windows, Unix, MacOS) and can be used to combine heterogeneous resources.

A group of connected machines forms a Condor pool. A self-sufficient Condor pool needs three types of machines: central manager, submit and execute. Any machine in the pool can perform any combination of these roles, however only one machine can be a central manager. A central manager keeps a record of all the connected resources. When a user submits a job to a Condor pool, it finds an empty resource satisfying the job requirements. Each resource publishes information regarding its capabilities such as processor speed, memory, operating system and any other user specified attributes, known as ClassAd [RLS98]. These ClassAds are used to identify the resources which match the job requirements. A submit machine allows the user to provide Condor with a job description specifying the job requirements. An execute machine when idle, processes jobs

submitted to a Condor pool. Many Condor pools can be configured to share loads using a flocking mechanism [ELvD*96] as long as the central managers of the pools can communicate with each other. This is useful for sharing across multiple departments of an organisation, where each department may have its own pool.

Condor monitors the user input and CPU load to determine idle resources and the administrators can configure the thresholds for implementing the desired job execution policies. This allows the donor to control the interference that a job may have on the resource. In case the donor withdraws the resource, the job is migrated to another vacant machine. To prevent loss of computations due to failures, Condor can create system level checkpoints of an executing job by saving the image of a running process [LTBL97]. However, there are some restrictions on jobs which can be checkpointed using this mechanism.

The MW tool [GKYL01] of Condor provides an API for developing and executing master-worker applications in an opportunistic environment. The API is a broker between the tasks and idle workers. It combines together the user application and the Condor resource management and message passing libraries. The user application specifies the list of tasks to be performed and the variation of resources is handled by the underlying Condor mechanisms.

3.7 Parallel Rendering on Grid

The current state-of-the-art parallel rendering algorithms presented in Section 2.9 are not suited for the grid since they are designed for dedicated resources. They rely on frequent communication between the rendering processes which may not be always possible on a grid. The parallel computations for rendering are tightly-coupled to achieve interactivity on such dedicated systems with fast communication links. In contrast, the compute nodes on the grid may reside on different networks with low bandwidth and high latencies as described earlier in Section 3.3.2. Hence, fault-tolerant parallel rendering algorithms with minimum reliance on communication need to be developed to effectively use grid computing.

Chong et al. [CSL06] presented a system for rendering animations on a computational grid. Their focus was to develop a lossless compression algorithm for transferring data between the nodes and they relied on the fault-tolerance

provided by the grid middleware. They created a portal for an easy access of underlying grid services to the users. Gooding et al. [GAST06] described an implementation of a distributed rendering environment for computational grid while Patoli [PGAB*09] et al. presented a preliminary study on creating a render farm based on a LAN-based desktop grid. Recently, Gonzalez-Morcillo et al. [GMWV*10] used a multi-agent architecture for decentralised rendering which employed importance maps to decide the workload distribution. The previous work for enabling high-fidelity rendering on the grid has not focused on optimising performance in the presence of faults to efficiently employ the enormous computational power offered by the grid.

Many scientific visualisation algorithms have been modified for parallel rendering using grid computing. A survey of such techniques can be found in [Mel08, BBC*05]. However, these techniques employ traditional fault-tolerance strategies (see Section 3.4.1) which require redundant computations, thereby decreasing system performance. Research in this area has focused on reducing the impact of such strategies by minimising redundant computations, for example [ZA10, GLH*08]. Hence, there is a need for developing alternate fault-tolerant mechanisms, which do not impede performance, by completely avoiding any redundant computations.

3.8 Summary

This chapter has described the theoretical concepts for computing on shared resources. A characterisation of such resources was presented and the differences between their various types were elucidated. Traditional fault-tolerance techniques, master-worker paradigm, and the grid middleware employed in this thesis, have also been described. A critical discussion on parallel rendering on the grid was also provided. The next chapter will present a primer on quasi-random sequences and explain how they can be used for pixel ordering.

CHAPTER 4

Quasi-random Sequences

Quasi-random numbers aim to bridge the gap between random numbers and regular grids. The idea is to adaptively fill the sampling space uniformly in a manner similar to regular grids while maintaining certain properties of random numbers. In fact, the term quasi-random is a misnomer [PTVF07] since there is nothing random about these numbers; they are completely deterministic. They have been designed such that each additional sample is chosen to fill the sampling space more uniformly avoiding clustering with previously generated samples. For this reason, they are also known as low-discrepancy sequences (see Section 4.1). Quasi-random sequences form the backbone of the algorithms presented in this thesis and this chapter serves as a brief primer about them.

4.1 Discrepancy

The discrepancy of a given sequence measures its deviation from an ideal uniform distribution. For a one dimensional sequence $x_1, \ldots, x_N$ it is defined as [KN74]:

$$D_N = D_N(x_1, \ldots, x_N) = \sup_{0 \leq \alpha < \beta \leq 1} \left| \frac{A([\alpha, \beta); N)}{N} - (\beta - \alpha) \right|$$

where $A([\alpha, \beta); N)$ is the counting function which is defined as the number of terms $x_i, 1 \leq i \leq N$, for which $x_i \in [\alpha, \beta)$, given a positive integer N and $[\alpha, \beta) \subset I$, where $I = [0, 1)$. A special form of discrepancy, known as star discrepancy, D_N^* is defined as:

$$D_N^* = D_N^*(x_1, \ldots, x_n) = \sup_{0 < \alpha \leq 1} \left| \frac{A([0, \alpha); N)}{N} - (\alpha) \right|$$

These can be extended to a multi-dimensional sequence, $\mathbf{x}_1 \ldots, \mathbf{x}_N$ as:

$$D_N = D_N(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sup_J \left| \frac{A(J;N)}{N} - \lambda(J) \right|$$

$$D_N^* = D_N^*(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sup_{J^*} \left| \frac{A(J^*;N)}{N} - \lambda(J^*) \right|$$

where J iterates through subintervals of I^k such that $J = \{(x_1, \ldots, x_k) \in I^k : \alpha_i \leq x_i \leq \beta_i \text{ for } 1 \leq i \leq k\}$ and $J^* = \{(x_1, \ldots, x_k) \in I^k : 0 \leq x_i < \alpha_i \text{ for } 1 \leq i \leq k\}$. Here λ represents the k-dimensional Lebesgue measure. The smaller the discrepancy of a sequence, the better the spacing between the samples. A $d-$dimensional sequence with N points satisfying the following criterion:

$$D_N^* \leq C_d \frac{(\ln N)^d}{N}$$

is termed as a quasi-random sequence, where C_d is a constant dependent on d only.

4.2 Types of Quasi-random Sequences

Numerous quasi-random sequences have been proposed, for example van der Corput [vdC35], Halton [Hal60], Hammersley [Ham60], Faure [Fau82], Sobol [Sob67] and Niederreiter [Nie88]. The following subsections discuss two sequences: van der Corput and Sobol which have been employed in this thesis.

Number	Binary Representation	Generation	van der Corput Sequence
0	$(0.0)_2$	$(0.0)_2$	0.0
1	$(1.0)_2$	$(0.1)_2$	0.5
2	$(10.0)_2$	$(0.01)_2$	0.25
3	$(11.0)_2$	$(0.11)_2$	0.75
4	$(100.0)_2$	$(0.001)_2$	0.125
5	$(101.0)_2$	$(0.101)_2$	0.625
6	$(110.0)_2$	$(0.011)_2$	0.375
7	$(111.0)_2$	$(0.111)_2$	0.875
8	$(1000.0)_2$	$(0.0001)_2$	0.0625
9	$(1001.0)_2$	$(0.1001)_2$	0.5625

Table 4.1: Generation of first 10 numbers of a base-2 van der Corput sequence

4.2.1 van der Corput Sequence

In 1935, a Dutch mathematician, J.G. van der Corput introduced a one dimensional quasi-random sequence defined over the interval $[0, 1)$ [vdC35]. A van der Corput sequence in base-b is generated by reversing the digits of natural numbers when represented in the base b, where $b \geq 2$. Table 4.1 shows the the generation of first 10 numbers of a base-2 van der Corput sequence. A finer mesh is generated for subdividing the $[0, 1)$ interval as the number of digits increase in the base-b representation.

Mathematically, a natural number, $n \in \mathbb{N}_0$, can be represented in base-b as (following [SS08]):

$$n = \sum_{j=1}^{m} a_j b^{j-1}$$

where $a_j \in \{0, ..., b-1\}$ and $b^{m-1} \leq n < b^m$, that is:

$$m = \left\lfloor \frac{\log n}{\log b} \right\rfloor + 1$$

The radical inverse function, $\phi_b(n) : \mathbb{N}_0 \to [0, 1)$, which is used for generating the van der Corput sequence is defined as:

$$\phi_b(n) = \sum_{j=1}^{\infty} \frac{a_j}{b^j}$$

A base-2 van der Corput sequence is the first dimension of a Halton sequence. Multi-dimensional Halton sequences are constructed in the same fashion as a van der Corput sequence, the difference being that each dimension uses a different prime number as its base in order to eliminate any correlation between the dimensions leading to a structure. Hence, in a Halton sequence the first dimension is generated using base-2, the second using base-3, the third using base-5 and so on. Faure sequences on the other hand, use the smallest prime number equal to or greater than the number of dimensions as the base for all the dimensions and then permute them for generating the sequence. The disadvantage of using these quasi-random sequences for higher dimensions stems from the fact that they need larger bases which increases the length of the cycle for covering the sample space. Therefore, a Sobol sequence is usually employed with higher dimensions as it uses base-2 for all dimensions as explained in the following section.

4.2.2 Sobol Sequence

Sobol sequence is a type of quasi-random sequence which generates samples over the interval $[0, 1)$ by uniformly partitioning the interval progressively using base-2 and reordering the samples. The mathematical theory behind construction of these sequences was published by Ilya Sobol [Sob67]. The generation process of a Sobol sequence is briefly described here based on the explanation given by Bratley and Fox [BF88].

The generation of a one dimension sequence, $x_i \in [0, 1)$ is considered such that the sequence fills the unit interval with low discrepancy . For this, a set of direction numbers v_i and a primitive polynomial P [Knu81] of degree d in field $\mathbb{Z}_2$ are needed. The direction number v_i, is a binary fraction represented as:

$$v_i = (0.v_{i1}v_{i2}v_{i3}\ldots)_2$$

where v_{ij} denotes the jth bit after the radix point in the binary representation of v_i. It can be also represented as:

$$v_i = \frac{m_i}{2^i} \tag{4.1}$$

where m_i is an odd integer such that $0 < m_i < 2^i$. The primitive polynomial P is given by:

$$P \equiv x^d + a_1x^{d-1} + \ldots + a_{d-1}x + 1$$

where $a_i \in \{0, 1\}$.

The choice of P is arbitrary and once it is selected a recurrence relationship can be defined to calculate m_i as:

$$m_i = 2a_1m_{i-1} \oplus 2^2a_2m_{i-2} \oplus \ldots \oplus 2^dm_{i-d} \oplus m_{i-d} \tag{4.2}$$

where $\oplus$ represents bitwise exclusive-or function. The choice of initial values $m_1, m_2, \ldots, m_d$ can be made freely such that constraints on m_i specified above are true.

Finally, the Sobol sequence can be generated using :

$$x_n = b_1v_1 \oplus b_2v_2 \oplus \ldots$$

where $(...b_3b_2b_1)_2$ is the binary representation of n. Antonov and Saleev [AS79]

enhanced the generation process by using Gray codes instead of the binary representation of n i.e.:

$$x_n = g_1 v_1 \oplus g_2 v_2 \oplus \dots \tag{4.3}$$

where $...g_3 g_2 g_1$ is the Gray code of n. As the Gray code for x_n and x_{n+1} differ by a single bit, the following recurrence relationship can be defined:

$$x_{n+1} = x_n \oplus v_c \tag{4.4}$$

where c is the index of the rightmost zero-bit in the binary representation of n.

Consider an example for generating a one dimensional Sobol sequence, where P is a primitive polynomial of degree 4 defined as:

$$P = x^4 + x + 1$$

The recurrence relationship for m_i can be calculated as explained above based on the chosen primitive polynomial using Equation(4.2). It is given by:

$$m_i = 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus 2^3 a_3 m_{i-3} \oplus 2^4 a_4 m_{i-4} \oplus m_{i-4} = 8m_{i-3} \oplus 16m_{i-4} \oplus m_{i-4} \tag{4.5}$$

To initialise the recurrence relationship, m_1 to m_4 need to be chosen based on the conditions defined above. Let $m_1 = 1, m_2 = 1, m_3 = 5, m_4 = 9$. Hence, $v_1 = 0.5, v_2 = 0.25, v_3 = 0.625, v_4 = 0.5625$ using Equation(4.1). The subsequent values of m_i and v_i can be calculated as follows using Equation(4.5):

$$\begin{aligned}
m_5 &= 8m_2 \oplus 16m_1 \oplus m_1 \quad = 8 \oplus 16 \oplus 1 = (1000)_2 \oplus (10000)_2 \oplus (1)_2 \\
&= (11001)_2 = 25 \\
v_5 &= \frac{25}{2^5} = 0.78125 \\
m_6 &= 8m_3 \oplus 16m_2 \oplus m_2 \quad = 40 \oplus 16 \oplus 1 = (10100)_2 \oplus (10000)_2 \oplus (1)_2 \\
&= (111001)_2 = 57 \\
v_6 &= \frac{57}{2^6} = 0.890625 \\
m_7 &= 8m_4 \oplus 16m_3 \oplus m_3 \quad = 72 \oplus 80 \oplus 5 = (1001000)_2 \oplus (1010000)_2 \oplus (101)_2 \\
&= (11101)_2 = 29 \\
v_7 &= \frac{29}{2^7} = 0.2265625
\end{aligned}$$

and so on.

The resulting Sobol sequence can be generated using Equation(4.4) as:

$$
\begin{array}{lrrcl}
\text{Initialisation} & : & x_0 & = & 0 \\
 & & n & = & (0)_2 \\
 & \therefore & c & = & 1 \\
\text{Step 1} & : & x_1 & = & x_0 \oplus v_1 \\
 & & & = & 0 \oplus 0.5 \\
 & & & = & 0.5 \\
 & & n & = & (1)_2 \\
 & \therefore & c & = & 2 \\
\text{Step 2} & : & x_2 & = & x_1 \oplus v_2 \\
 & & & = & 0.5 \oplus 0.25 \\
 & & & = & 0.25 \\
 & & n & = & (10)_2 \\
 & \therefore & c & = & 1 \\
\text{Step 3} & : & x_3 & = & x_2 \oplus v_1 \\
 & & & = & 0.25 \oplus 0.5 \\
 & & & = & 0.75 \\
 & & n & = & (11)_2 \\
 & \therefore & c & = & 3 \\
\text{Step 4} & : & x_4 & = & x_3 \oplus v_3 \\
 & & & = & 0.75 \oplus 0.625 \\
 & & & = & 0.375
\end{array}
$$

and so on. Instead of using the recursion, x_n for any value of n can be calculated using Equation(4.3). For example, let $n = 50$:

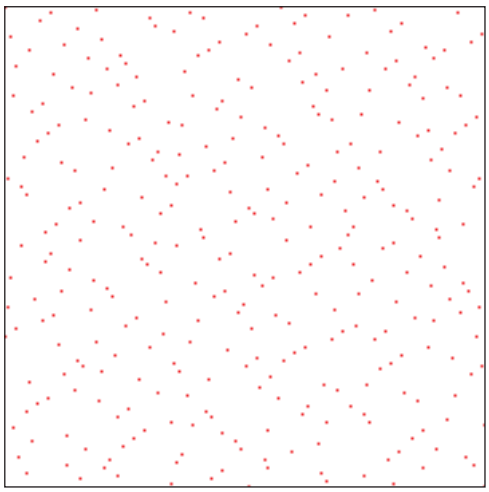
$$
\begin{aligned}
n &= (110010)_2 \\
\text{Gray Code}(n) &= (101011)_2 \\
x_{50} &= v_6 \oplus v_4 \oplus v_2 \oplus v_1 \\
&= 0.890625 \oplus 0.5625 \oplus 0.25 \oplus 0.5 \\
&= (0.111001)_2 \oplus (0.1001)_2 \oplus (0.01)_2 \oplus (0.1)_2 \\
&= (0.101101)_2 \\
&= 0.703125
\end{aligned}
$$

Sobol [Sob67] showed that for generating a d-dimensional sequence, d different primitive polynomials are required to calculate d sets of direction numbers. Each set of direction numbers is used to generate a different component of the multi-dimensional sequence. Sobol sequences usually perform better than Faure and Halton sequences for higher dimensions as they use base-2 for all dimensions, but they are prone to multi-dimensional clustering due to correlations between the dimensions.

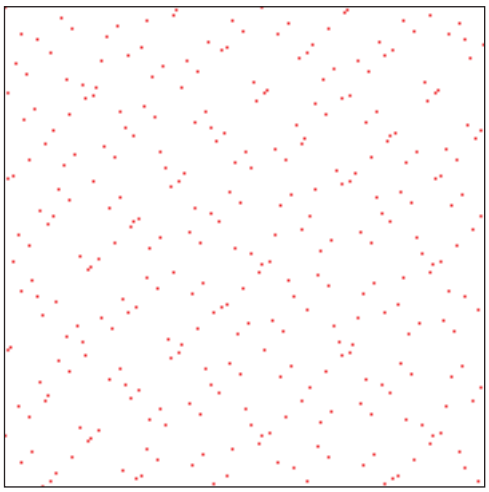
4.3 Scaling and Quantisation of Quasi-random Sequences

In this thesis, quasi-random sampling has been used for selection and ordering of pixels from images as it leads to a good coverage of the sampling space. Quasi-random sequences typically generate samples in the $[0, 1)$ interval. Hence, these samples need to be scaled and quantised for selecting the appropriate pixels. A carefully chosen scaling factor is required such that the quantisation leads to complete coverage of the sampling space. For example, consider ordering of five of numbers (0 to 4) using base-2 van der Corput sequence. The first five samples of the sequence are $\{0, 0.5, 0.25, 0.75, 0.125\}$ which if scaled by a factor of five lead to $\{0, 2.5, 1.25, 3.75, 0.625\}$. The quantisation obtained by rounding-off the scaled samples would be $\{0, 2, 1, 4, 1\}$ which leads to improper coverage, i.e. duplicate and missing samples. However, if a scaling factor s is chosen such that $s > b^n$, where b is the base used for generation and n is the smallest integer satisfying the condition, it leads to a better scaling. Although, now s samples are needed. For the current example, $s = 8$ and the scaled samples become $\{0, 4, 2, 6, 1, 5, 3, 7\}$. The quantisation is straightforward but more samples are required to cover the sample space and some of them need to be discarded. Finally, a base-2 van der Corput sequence can be used to order 5 samples as $\{0, 4, 2, 1, 3\}$. This concept can be used for scaling and quantisation of the samples generated by any quasi-random sequence. Scaling and quantisation can be extended for multi-dimensional sampling, where each dimension can be scaled with a different scaling factor.

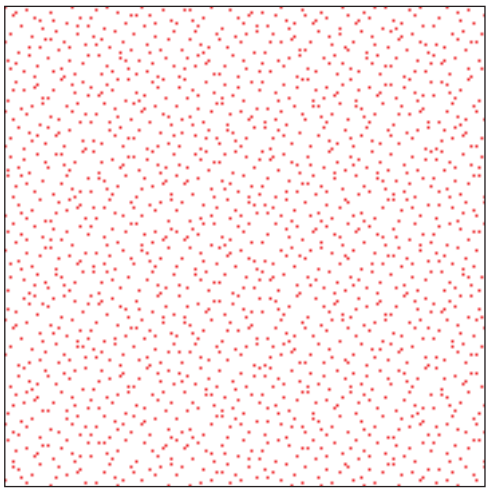
Figure 4.1 depicts progressive sampling of an image with varying number of samples using Halton and Sobol sequences (pixels sampled once are coloured red, ones which are sampled more than once are coloured blue and unsampled ones are white). It can be seen that addition of samples result in a better coverage of



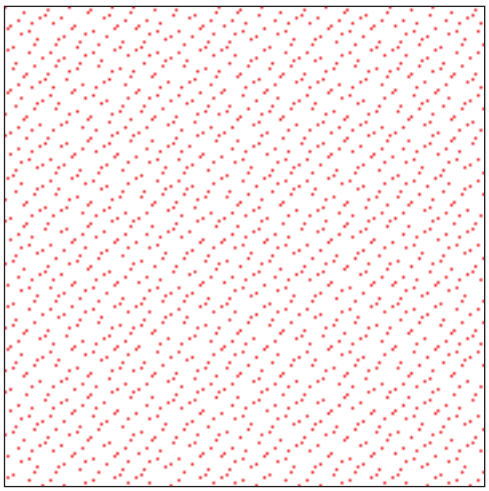
(a) 1% Halton



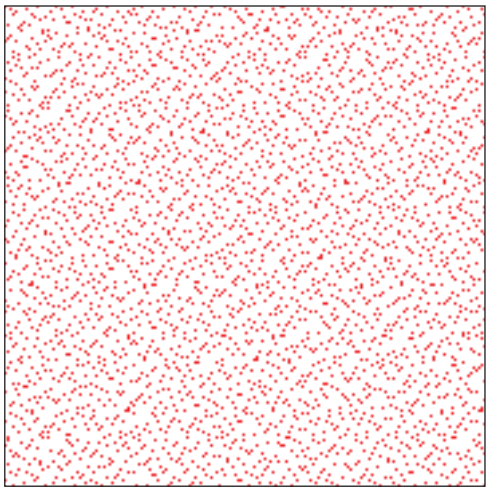
(b) 1% Sobol



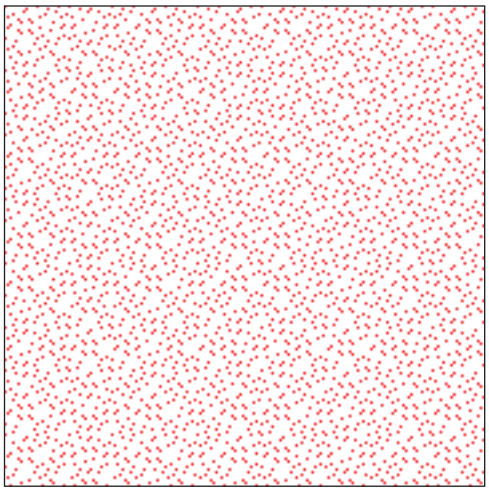
(c) 5% Halton



(d) 5% Sobol

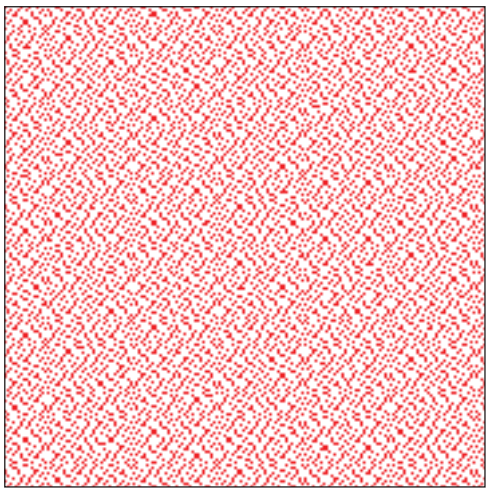


(e) 10% Halton

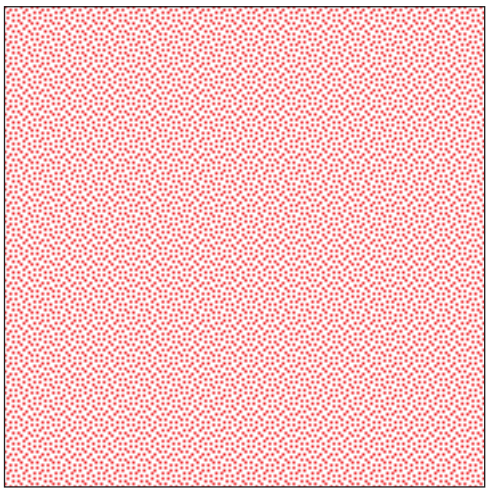


(f) 10% Sobol

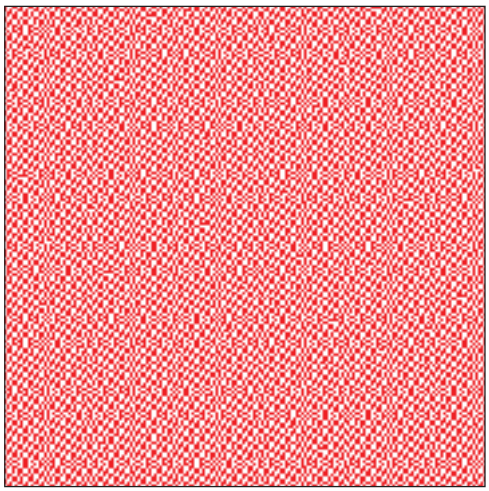
Figure 4.1: Two-dimensional Halton and Sobol sequences with varying number of samples



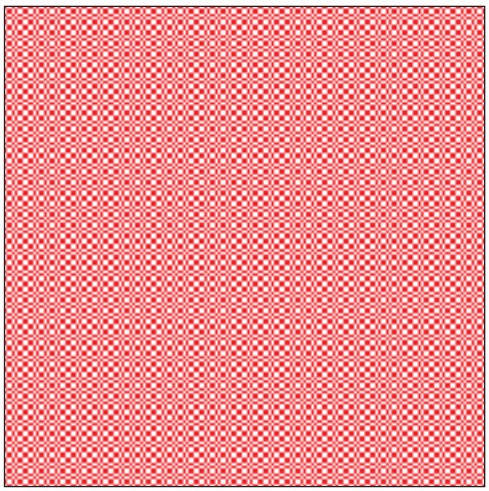
(g) 25% Halton



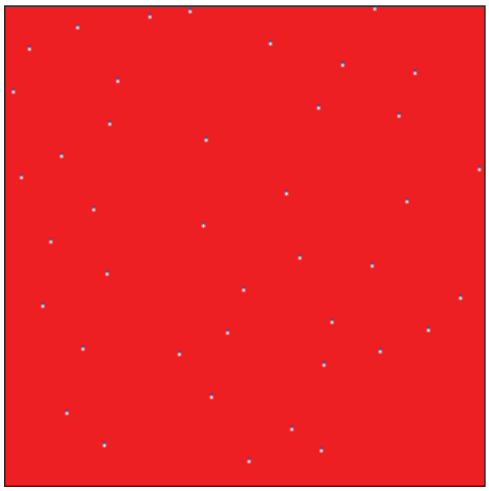
(h) 25% Sobol



(i) 50% Halton



(j) 50% Sobol



(k) 100% Halton

(l) 100% Sobol

Figure 4.1: Two-dimensional Halton and Sobol sequences with varying number of samples

the whole image. Figure 4.1k shows a few pixels which have been sampled more than once and some which have not been sampled after 100% of the samples have been generated. A two-dimensional Halton sequence uses base-3 for one of its dimensions and hence precision errors occur on a modern day CPU which performs the calculations in base-2. This results in the quantisation error as shown in Figure 4.1k. In order to avoid such errors, quasi-random sequences with base-2 should be used for a complete coverage. This thesis uses the Sobol and van der Corput implementations for generating base-2 sequences presented in [KK02] for two-dimensional sampling, where each of the two sequences samples a different dimension. This implementation allows different base-2 sequences to be generated resulting in different sets of pixels by randomised digit scrambling rather than a deterministic sequence [KK02]. It helps in improving the fault-tolerance by grouping different sets of pixels together as a job rather than using a fixed group.

4.4 Summary

This chapter provided an overview of quasi-random sequences. The concept of scaling and quantisation of such sequences was also covered since it will be used for pixel ordering in the algorithms presented in the next chapters. The importance of using a base-2 quasi-random sequence for achieving a complete coverage was also illustrated. This will enable a fault-tolerance strategy to be developed for image subdivision, as explained further in Section 6.1.

CHAPTER 5

Animation Rendering on Computational Grids

Generation of high-fidelity animations is a computationally expensive process, often taking many hours to complete (see Section 2.6.1). Massive parallelism is possible using a computational grid (henceforth referred to as grid in this chapter, unless specified otherwise). However, since it is a multi-user environment with a large number of nodes potentially separated by substantial network distances (see Section 3.2); communication should be kept to a minimum. While for some rendering algorithms, computing animations on the grid may be a simple task of assigning an individual frame for each processor, certain acceleration data structures, such as irradiance caching (see Section 2.6.3), require different approaches. The irradiance cache may be used to significantly reduce the computational requirements when generating high-fidelity animations. Parallel solutions for irradiance caching using shared memory or message passing (see Section 2.9.1) are not ideal for grid computing due to the communication overhead and must be adapted for this highly parallel environment.

This chapter is based on [ACD08] and it is organised as follows: Section 5.1 introduces the chapter. Section 5.2 discusses the issues related to rendering on the grid and presents a two-pass approach to overcome these issues. Section 5.3 contains the results of the two-pass approach. A discussion on the grid is provided in Section 5.4 and finally the chapter is summarised in Section 5.5.

5.1 Introduction

A significant amount of work has been done on parallel rendering (see Section 2.9) but little has been done to enable it on the grid. The algorithms used to render on the grid should be designed such that there is minimum sharing of data between the nodes to fully utilise the available resources. The initial results of high-fidelity rendering on the grid demonstrate its computational potential.

This chapter presents rendering of high-fidelity walk-through animations using a two-pass approach by adapting the irradiance cache algorithm for parallel rendering using grid computing. This approach exploits the temporal coherence between animation frames to gain significant speed-up and enhance the visual quality. The key feature of this approach is that it does not use any additional data structure and can thus be used with any irradiance cache or similar acceleration mechanism for rendering on the grid. Most of the extensions of irradiance cache mentioned in Section 2.6.3, could benefit from the approach presented in this chapter via a similar straightforward conversion.

When rendering animation frames in parallel, artefacts are generated due to temporal incoherence if the irradiance cache is not shared between the rendering processes. The artefacts are a result of interpolation of irradiance values from different irradiance caches on close parts of the model appearing in different frames of the animation. The proposed method uses a two-pass approach for rendering the animations as suggested by [LS98]. The irradiance at selected points along the animation is computed in the initial pass, while the second pass computes the animation in the traditional way. Any visual temporal noise in the animations is eliminated without significantly increasing the communication costs and a speed-up is obtained due to the saving on recomputation of data while rendering high-fidelity animations on the grid.

5.2 Rendering on the grid

This section discusses two approaches for rendering of high-fidelity walk-through animations on the grid using irradiance cache.

5.2.1 Single-pass Approach

A straightforward approach for rendering on the grid would be to submit each frame of the animation to the grid as an independent job and once all frames have been rendered the animation could be compiled [CSL06]. A problem with this approach is the noise artefacts which are generated due to lack of sharing of data between the frames. This is seen as temporal noise (flickering and shimmering) in the compiled animation and may draw viewer's visual attention towards unimportant parts of the animation. Henceforth this approach is referred to as the single-pass approach.

In the single-pass approach, each frame is calculated independently on a different machine on the grid. Therefore, there is no sharing of irradiance cache samples between machines calculating successive frames. This results in interpolation of irradiance values at same (or relatively close) points in the different frames using different irradiance cache samples (see Figure 5.1). Theoretically, this problem can be solved if the search radius for the cache is kept very small. But practically, it is not possible to keep the radius very small for rendering in reasonable times. Creating a system to share irradiance cache data between different nodes on the grid while the frames are being computed would incur large communication cost as different nodes on the grid may be on different clusters separated by large network distances. Since the grid is a massively parallel system there would be many copies of the irradiance cache distributed among different nodes and it would be difficult to update each cache with the values calculated by the other nodes in real-time without stalling the rendering process.

Furthermore, the computation of frames is usually slower while rendering using a single-pass approach. The irradiance values calculated on one part of the model in one frame can be reused to interpolate irradiance in successive frames. But there is no sharing of the irradiance cache between the nodes, hence computations done on one node cannot be reused for computation on other nodes resulting in recomputation of similar values. This would result in extra computation leading to poorer execution times.

5.2.2 Two-pass Approach

To overcome the problems discussed in the previous section, a two-pass approach is proposed for rendering on grid systems. This approach considerably reduces visual temporal noise in animations and speeds up the process taking advantage

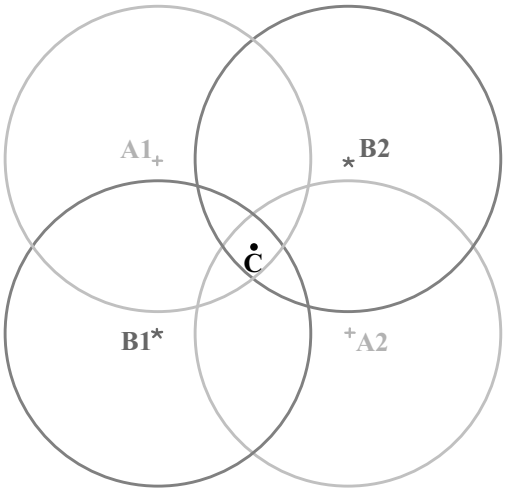


Figure 5.1: The irradiance value at point C may be interpolated in one frame of the animation from points A1 and A2, while points B1 and B2 may be used for interpolation in another frame. As the two frames are rendered on different nodes on the grid, irradiance value at C would be different in the two frames giving rise to temporal noise in the animation.

of coherence between successive frames. The two passes can be summarised as:

1. A set of frames is selected from all the frames to be rendered, by giving equal weight to change in direction and position of camera. These selected frames are rendered in order to generate separate irradiance caches. The caches thus obtained are then merged together on one node.

2. The merged cache is distributed to the assigned computational nodes on the grid. The scene is then rendered using the merged irradiance cache, queuing each frame as a job on the grid.

5.2.2.1 The First Pass

The first pass of the two-pass approach consists of rendering selected frames so as to generate the irradiance cache. Selection of frames needs to be done keeping in mind that the scene geometry is sampled in a way that most of the irradiance values in the second pass are interpolated from the irradiance cache generated in the first pass, rather than calculated. If the frames are not selected wisely, scene geometry may not be sampled properly leading to recomputation of irradiance cache samples on different nodes on the grid. This would result in temporal noise and reduce the performance that can be achieved with a properly sampled

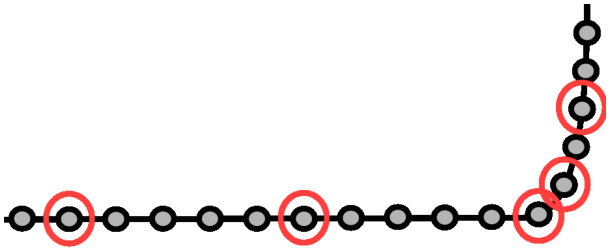


Figure 5.2: The encircled frames are selected for the first pass, chosen by giving equal weight to both change in direction and position of the camera.

geometry. It is possible to sample the geometry directly, but it is simpler to choose a frame from a set of frames.

Knowledge of the camera path allows employment of a simple heuristic to identify the frames, most viable for generating separate caches. The normalised change of angles (α) and normalised distance travelled by the camera (β) between successive frames is summed (γ). The normalisation ensures an equal importance between α and β. The value of γ helps in determining the greatest level of change between frames. This is used to select N equally spaced frames based on cumulative change of γ where N is the number of idle resources available on the grid. The value of N is an approximate value observed at the start of the rendering process and differs considerably at and during the runtime since the grid is a dynamic environment with multiple users. The frame selection process is depicted in Figure 5.2.

Instead of calculating the complete frames, a number of pixels are rendered until an irradiance cache hit/miss threshold is reached. Since the number of rays needed to be shot is not known until execution, a Sobol sequence (see Section 4.2.2) is used to generate the pixels from which to calculate the rays. This quasi-random sampling helps in shooting rays such that they are reasonably well-distributed over the parts of the geometry required by the animation. The rendering of each of the selected frames is queued up as a job on the grid. The computation is stopped once the number of cache hits to cache misses is 10 to 1. This threshold is checked only after a user-defined base amount of rays (typically 128, but this could be modified based on scene complexity) has been calculated. The various irradiance cache data obtained at each node is then merged by a single node. This merged cache is distributed to the nodes on the grid in the second pass to compute the complete animation.

5.2.2.2 The Second Pass

In this pass, each frame of the animation being rendered is submitted to the grid as a different job. The frames are calculated now with the help of the merged irradiance cache. Newly created samples are still cached but are not shared. The resulting frames can be stored on a data server using the data service of the grid or they can be sent back to the user's machine. Job failure is detected by the underlying grid technologies and resubmitted. Once every frame has been computed, the animation is compiled.

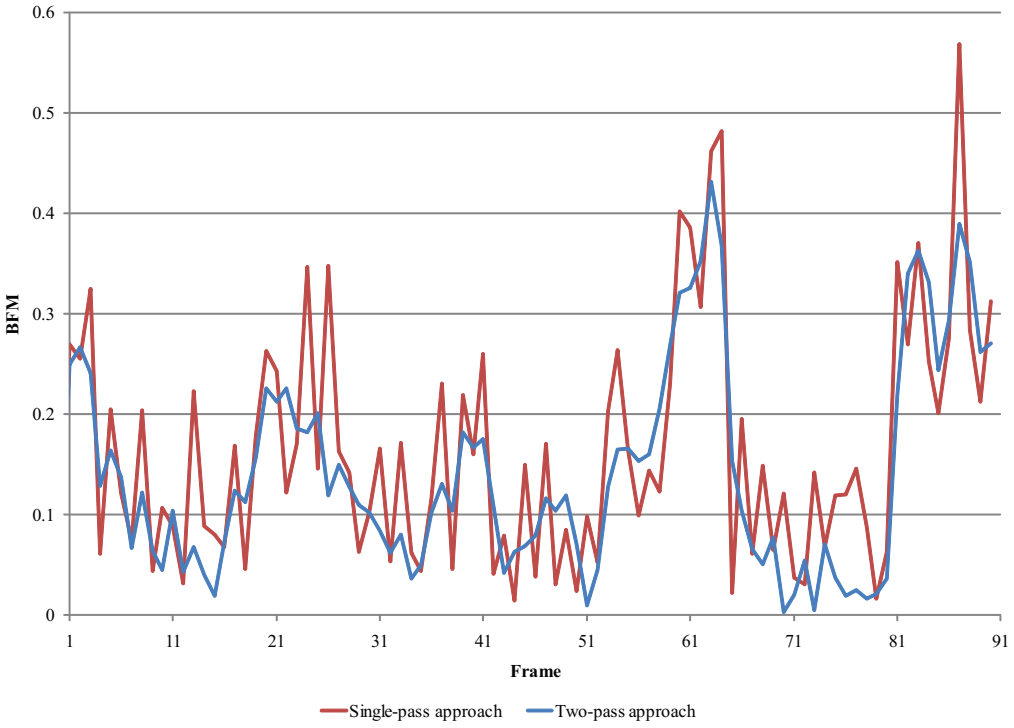
5.3 Results

Both the single-pass and two-pass approach described in the previous section have been implemented and tested on the National Grid Service (NGS) [NGS10] using the Globus toolkit (see Section 3.6.1). The Radiance Light Simulation package [LS98] has been modified to adapt to the two-pass approach. The modified binaries were transferred to the nodes for rendering frames where the animations were rendered using 2 to 3 indirect diffuse bounces as suggested by McNamara [McN05] and high quality parameters for the other settings.

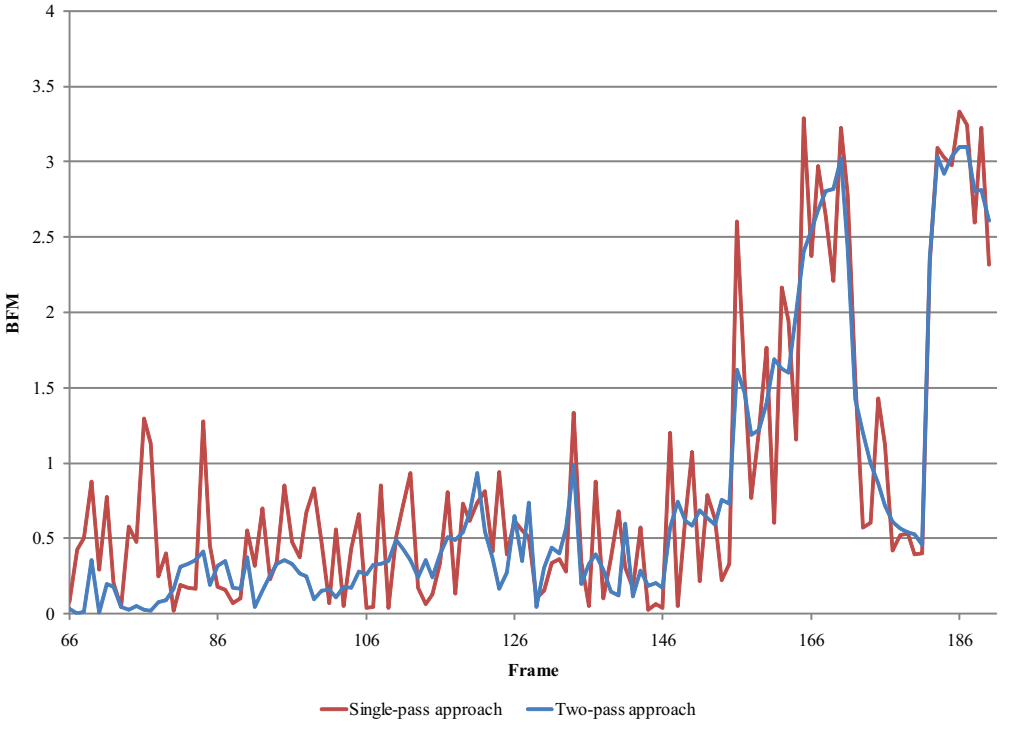
Approximately two hundred processors were used while rendering animations on the NGS grid, which had a total of about a thousand processors at its four core sites. The number of processors used for rendering was restricted by the multi-user environment on the grid.

5.3.1 Visual Quality

The Brightness Flickering Metric (BFM) [BFM10] was employed to compare the visual quality of animations generated by the single-pass approach and the two-pass approach. BFM is measured by calculating the modulus of difference of average brightness values between successive frames. The average BFM is a poor indicator of visual quality in comparing temporal noise since it considers the average brightness of the complete frame while the temporal noise is generally restricted to some areas of the animations. The flickering can be observed in the portions of the animations which are predominantly lit by indirect diffuse lighting. The BFM graphs shown in Figure 5.3 compare selected portions of the animations where the model is mainly illuminated by indirect diffuse lighting.

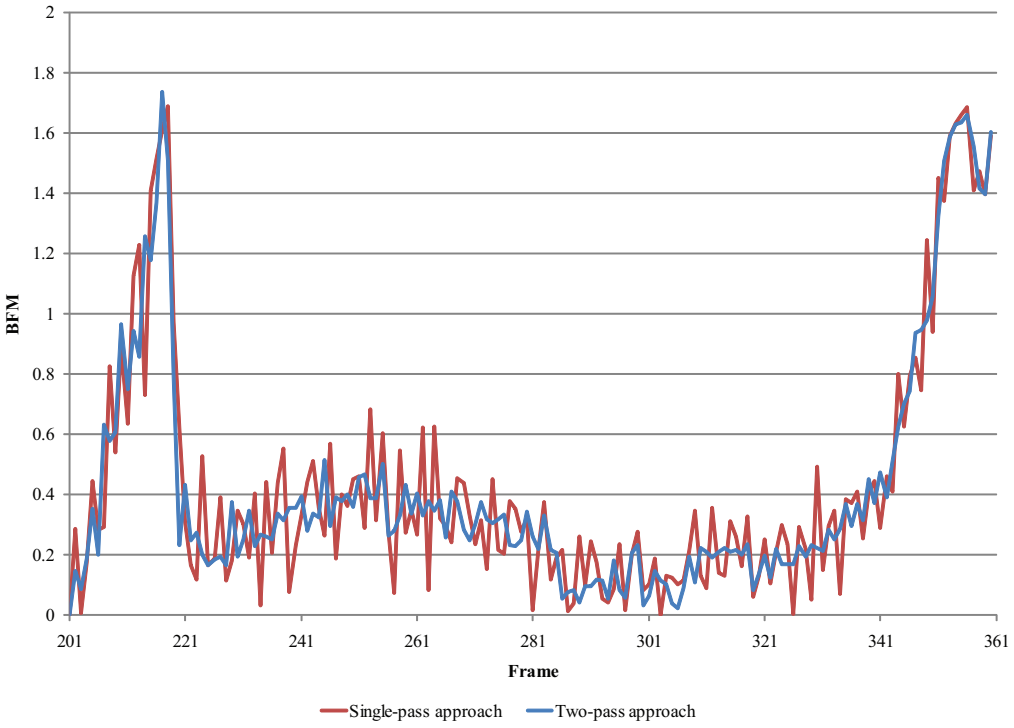


(a) Kalabsha Walk-through

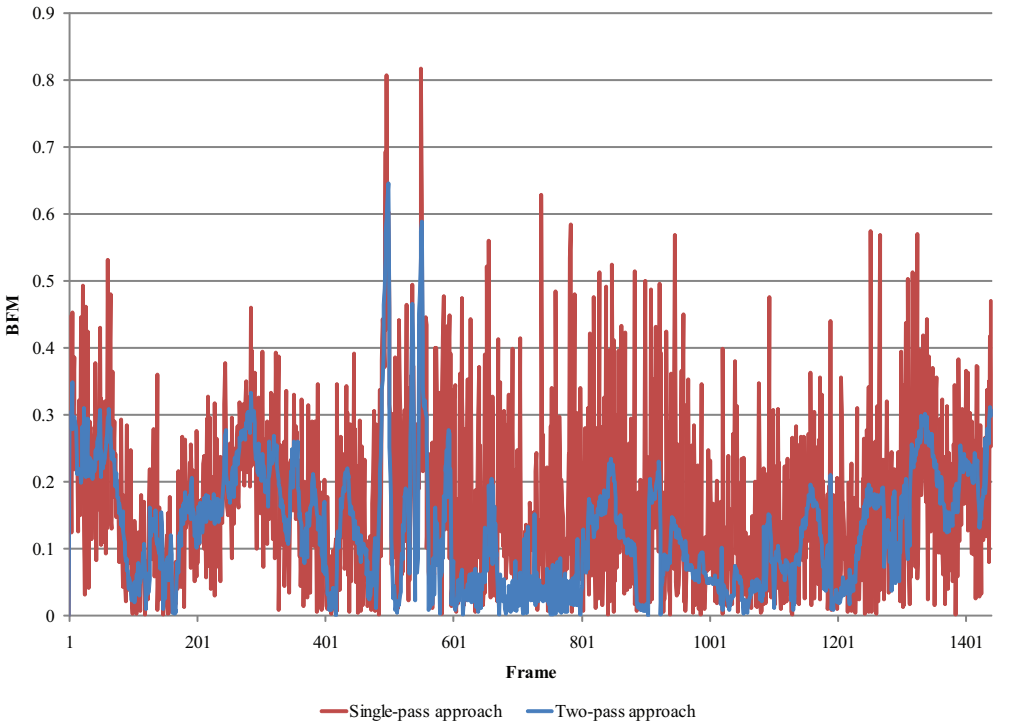


(b) Artgallery

Figure 5.3: BFM graphs comparing the two approaches. Kalabsha Walk-through animation was calculated on a single processor while the others were computed on the NGS.

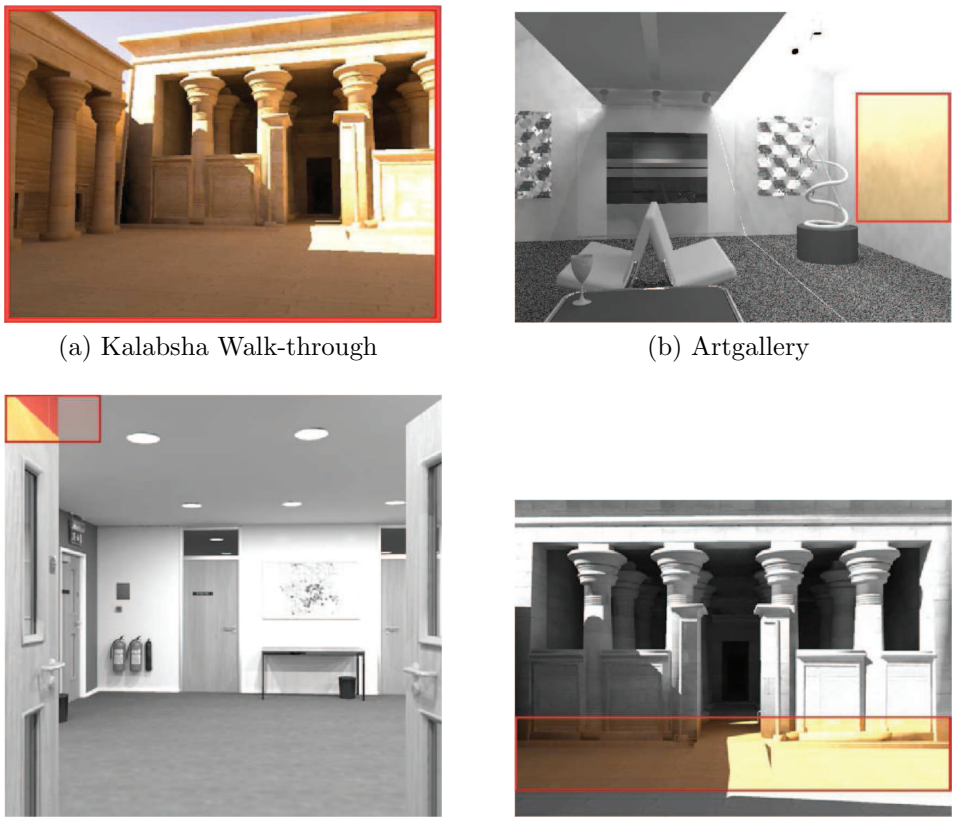


(c) Corridor



(d) Kalabsha Rotation

Figure 5.3: BFM graphs comparing the two approaches. Kalabsha Walk-through animation was calculated on a single processor while the others were computed on the NGS.



(a) Kalabsha Walk-through

(b) Artgallery

(c) Corridor

(d) Kalabsha Rotation

Figure 5.4: The coloured portion of the frame indicates the part of the animation chosen for calculating the BFM graphs.

As can be seen from the BFM graphs in Figure 5.3, the change of brightness in the single-pass approach follows the trend of the change in brightness in animation rendered using the two-pass approach, but the BFM graphs of animations from the single-pass approach have more overshoots, indicating flickering. The coloured portion of the frame in Figure 5.4 shows the portion of the animation chosen for obtaining the BFM graphs. For the Kalabsha Walk-through animation (see Figure 5.4a) most portions are lit by indirect diffuse lighting, hence the BFM graph is plotted for the complete animation.

5.3.2 Timing and Speed-up

The timing results illustrating the speed-up of the two-pass approach over the single-pass approach along with the description of the animations have been summarised in Table 5.1. The makespan is the time taken from the moment a

Animation Name	Frames	Resolution	Rendered on	Average Computation Time per frame (hr:min)	Makespan (hr:min)		Speed-up comparing Two-pass and Single-Pass Approach
					Two-pass Approach	Single-pass Approach	
Kalabsha Walk-through	91	1024×768	Intel Xeon 3.0 Ghz	00:06	02:03	09:37	4.69
Artgallery	192	2048×1536	NGS	00:37	04:35	05:49	1.26
Corridor	361	2048×2048	NGS	03:43	05:21	11:41	2.19
Kalabsha Rotation	1441	2048×1536	NGS	00:09	01:48	02:06	1.16

Table 5.1: Animation Description and Timings

script is submitted to the grid which submits the jobs till the time every frame has been rendered. Specifically, for the two-pass approach it includes the time taken for both the passes which are run successively on the grid by a script which first submits jobs for the first pass, then checks for the completion of the first pass, merges the cache and finally launches jobs for the second pass.

The comparison of timings while rendering on the grid with the timings on a single processor is infeasible due to the amount of time it takes to render the animations on a single processor. However, an average computation time of a single frame is presented as an overall indication. It can be seen from the Table 5.1, that the Kalabsha Rotation animation was rendered in less than two hours on the grid, which is a high resolution animation with 1441 frames. It would have taken more than two hundred hours approximately (average computation time per frame multiplied by number of frames), if the animation was rendered on a single processor. This was achieved only because of the massive parallelism offered by the grid. Similar results can be observed for the Art Gallery and Corridor animations.

The speed-up comparing the two-pass approach and the single-pass approach on a grid system is affected by many factors such as number of other users using the grid services, the network load on the grid interconnects, the number of nodes running on the grid. Furthermore, communication costs on the grid are high. These factors prevent measurement of the complete speed-up achieved by computing animations using the two-pass approach over the single-pass approach. Hence, a result for a small animation is presented, which was rendered on a single Intel Xeon 3.0 GHz processor (see Table 5.1) as an indicator of speed-up of the two-pass approach over single-pass approach. This is obtained due to saving on recomputation time. Also, it is greater than the speed-up achieved on the grid since additional factors are introduced while computing on the grid. In particular, the communication overhead of job submission and the time taken by the grid

middleware are the major factors affecting the speed-up on the grid.

Even though by using a two-pass approach increases the overheads of job submission and communication, it has still been able to achieve speed-up in all cases over the single-pass approach. This can be attributed mostly due to the saving of computation time by avoiding recomputation on different nodes of the grid.

5.4 Discussion

Rendering animations on the grid provided a practical insight into the current state of grid computing. The grid middleware has been designed for robust security mechanisms and is suited for long running jobs. The communication costs on the grid make it unsuitable for real-time rendering and small rendering jobs. The initial process required for gaining access and the setup time involved for installation of middleware may hinder new users from utilising the full potential of the grid. Although provisions have been made to create simple access mechanisms, they are restricted to popular software packages only. For executing custom code, the user needs a good knowledge of working with the grid middleware. Also, access to the NGS is usually restricted to UK academics and hence not everyone can apply for computation time. Furthermore, it is difficult to predict the availability of required resources due to their shared nature, and the user computations compete for access with jobs submitted by others. However, once the computation starts the resources are usually dedicated for that computation. The grid is an extremely useful resource for rendering high-fidelity animations for which the job computation time is high in comparison to the communication costs.

5.5 Summary

This chapter illustrates the use of grid computing for rendering high-fidelity animations in reasonable time. A two-pass irradiance caching algorithm was used for indirect light calculations, exploiting temporal coherence between animation frames. The results indicate that there is no visible temporal noise in animations rendered using two-pass approach, enhancing their visual quality. Using the two-pass approach, a speed-up over the single-pass approach was achieved despite the

fact that there are more communication overheads on the grid. This speed-up is attributed to substantial time saving by avoiding recomputation of irradiance cache data. Since no additional data structures have been used while adapting the irradiance cache algorithm for rendering of animations on the grid, this approach can be applied to most renderers which use irradiance caching and other similar algorithms based on it (such as those mentioned in the Section 2.6.3).

LAN-based desktop grids (henceforth referred to as desktop grids in this thesis) provide more interesting possibilities for computing at the cost of a more challenging environment for rendering due to the higher volatility of resources. A desktop grid can be formed by combining resources of an already existing infrastructure, and hence they are easily accessible to the users. However, novel fault-tolerant rendering algorithms are crucial for effective use of the desktop grids. The growing possibilities of desktop grid computing, lead to the investigation of their suitability for the purpose of high-fidelity rendering. In the next few chapters, algorithms for rendering on desktop grids will be discussed.

CHAPTER 6

Time-constrained Offline Rendering on Desktop Grids

This chapter presents fault-tolerant algorithms for rendering high-fidelity images on a desktop grid within a given time-constraint. Due to the dynamic nature of such resources, the task assignment does not rely on subdividing the image into tiles. Instead, a progressive approach is used that encompasses aspects of the entire image for each task and ensures that the time-constraints are met. Traditional reconstruction techniques are used to calculate the missing data. This approach is designed to avoid redundancy enabling time-constraint rendering. As a further enhancement, the algorithm decomposes the computation into components representing different tasks to achieve better visual quality considering the time-constraint and variable resources [Deb06]. The results illustrate how the component-based approach maintains a better visual fidelity considering a given time-constraint while making use of volatile computational resources.

This chapter is based on [ADD*09] and it is organised as follows: Section 6.1 introduces the chapter. Section 6.2 discusses the novel time-constraint fault-tolerant parallel rendering algorithms. Results are presented in Section 6.3 and finally the chapter is summarised in Section 6.4.

6.1 Introduction

Computational problems which can be modelled as bag-of-tasks applications are suitable for execution on desktop grids as they can be broken into smaller independent subtasks (see Section 3.3.2). These subtasks can be executed in parallel on any of the available resources of a desktop grid using a task pull-model

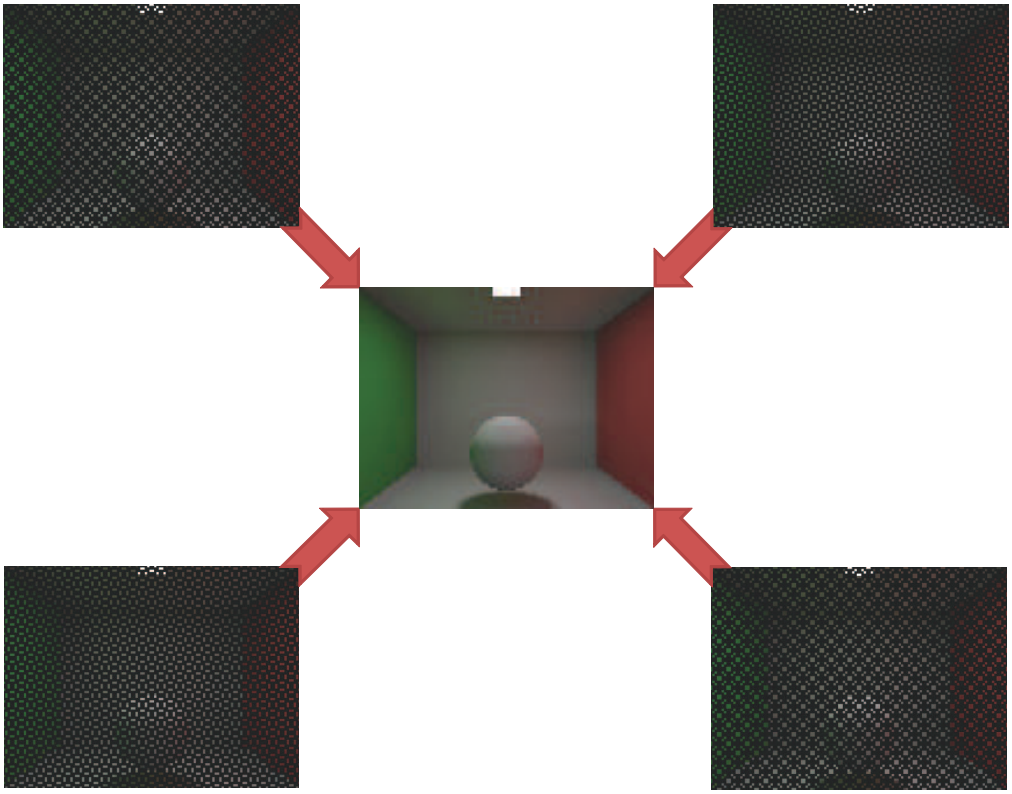


Figure 6.1: An image of the Cornell Box divided into 4 groups of quasi-randomly chosen pixels. Each of these groups can be independently computed in parallel. In practice, the image is divided into many more groups.

whereby any idle resource pulls off a task from a central server. Volatility of resources means that redundancy is needed for providing fault-tolerance. Ray tracing (see Section 2.5) easily lends itself to this model as computation of one pixel is completely independent of any other pixel. However, scheduling each pixel as a different task would incur a high communication overhead and therefore pixels are generally grouped together in the form of image tiles when parallel computing is employed.

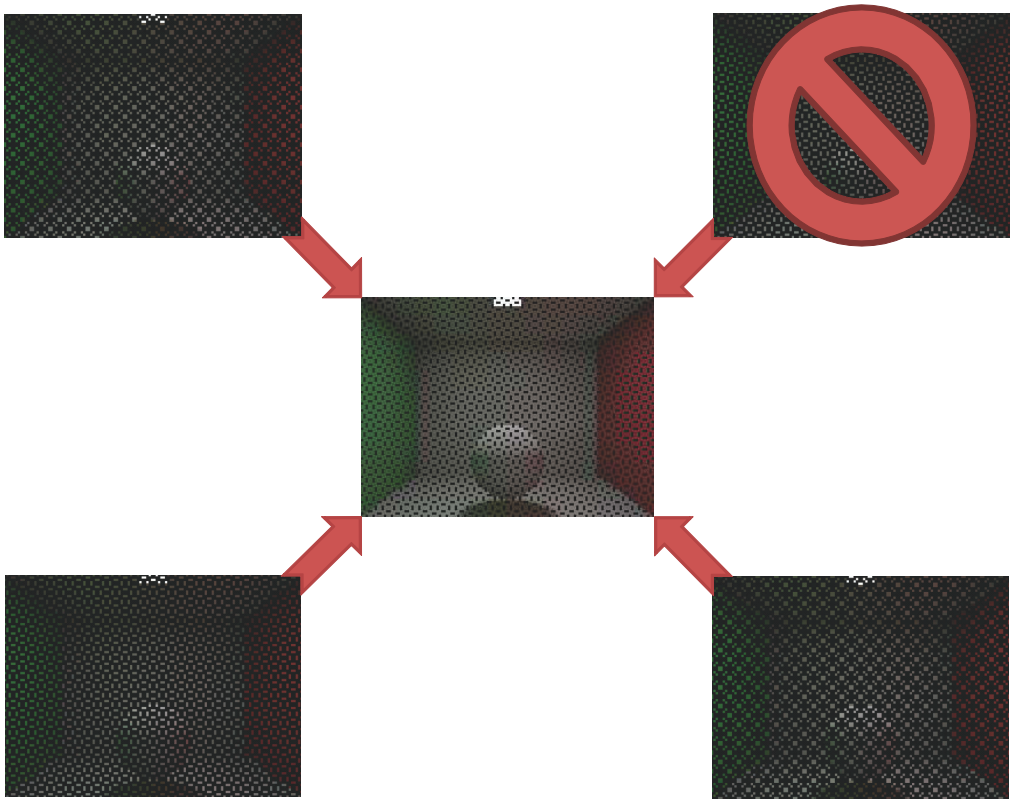
The use of time-constraints is an elegant way of employing changeable resources. However, such an approach introduces many challenges. Traditionally, an image is subdivided into tiles for parallel rendering such that each tile can be calculated in parallel [CDR02]. The image is composited when all the tiles are rendered. For desktop grids, redundancy could be employed to ensure all the tiles are computed (see Section 3.4.1.1). This would entail that if one of the tiles fails to come back (within a specified time) due to a delay or fault, a duplicate copy of the same job would be computed. However, redundancy is not ideal

when rendering towards a deadline as it can add substantially to the rendering time needed for a frame, thus hindering performance. Instead, an image can be subdivided into groups of pixels chosen quasi-randomly (see Chapter 4) over the complete image space instead of using tiles (see Figure 6.1). This enables a fair coverage of the whole image per job. If a job fails to complete, image reconstruction techniques are used to fill in the missing data, Figure 6.2a. It is difficult to reconstruct an image when a tile of pixels is grouped together as a job, since there would be significant holes in the case where a task fails without redundancy, as shown in Figure 6.2b. The advantage of using a quasi-random sequence over a purely random sequence is that it provides low discrepancy (fills the space more uniformly). A regular sampling pattern could, on the other hand lead to undesirable structured noise in comparison to quasi-random sampling in case of faults, as shown in Figure 6.3. By using image reconstruction algorithms, redundancy can be avoided for parallel rendering while maintaining time-constraints in a shared environment.

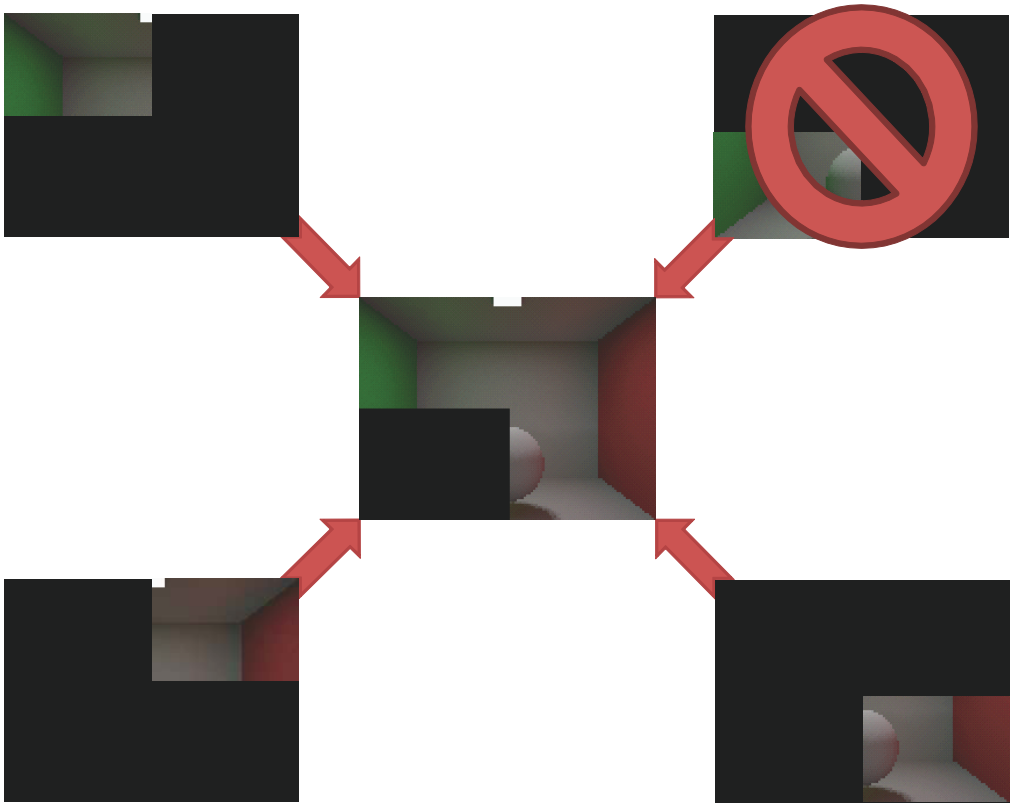
Generation of high-fidelity images using ray tracing requires multiple-per-pixel computations to account for global illumination effects. A further enhancement to the novel way of using time-variant resources such as a desktop grid is presented, by breaking down these per pixel computations into components. This enables better visual quality to be achieved within a user-defined time-constraint. In this component-based algorithm, direct and indirect lighting calculations are separated into different tasks. Furthermore, each task refines the solution progressively and consists of a set of quasi-randomly chosen pixels in order to avoid redundancy as explained above.

6.2 Time-constrained Fault-tolerant Parallel Rendering Algorithms

In this section, details of two novel methods for time-constrained fault-tolerant parallel rendering are discussed. The goal of the presented algorithms is to break down the rendering computations into subtasks in an elegant manner such that each discrete subtask refines the solution progressively while taking care that in the case where one or more of them fail to complete, the missing data can be reconstructed. The description of various attributes that help the presented algorithms to achieve this objective are addressed in the following subsections.

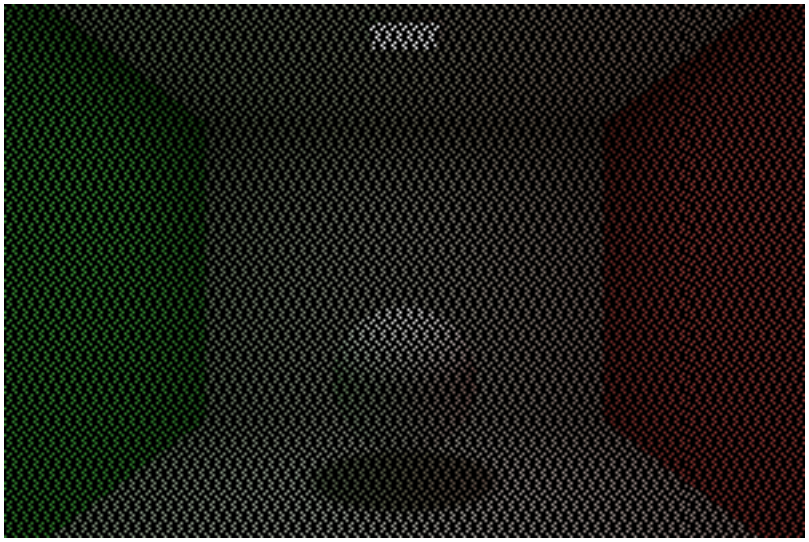


(a) Quasi-random Sampling

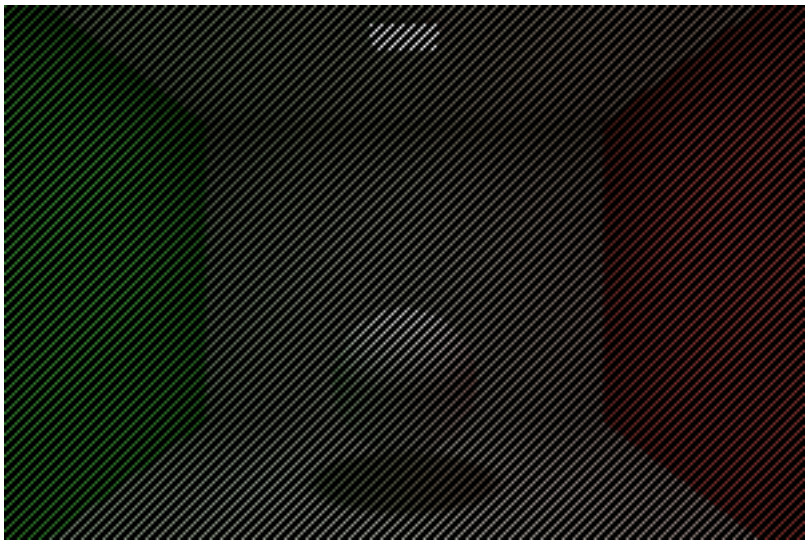


(b) Tile-based Sampling

Figure 6.2: Image subdivision techniques and the resultant image in case of faults.

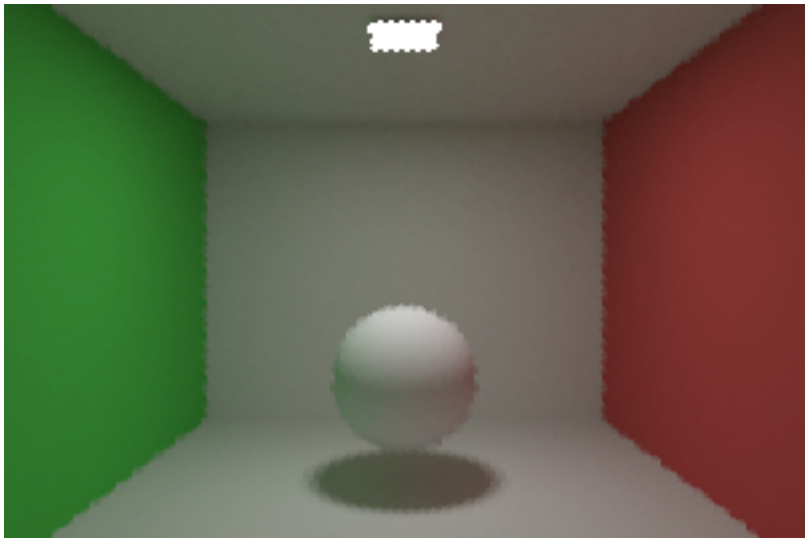


(a) Quasi-random sampling

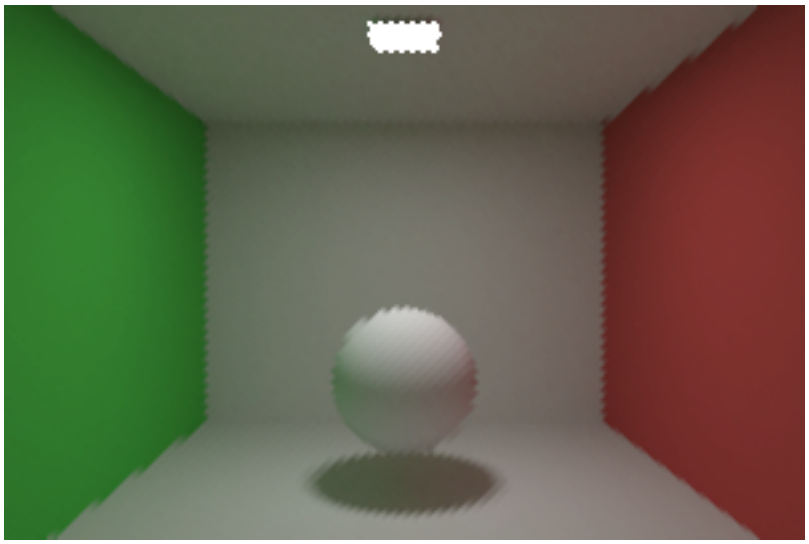


(b) Regular sampling

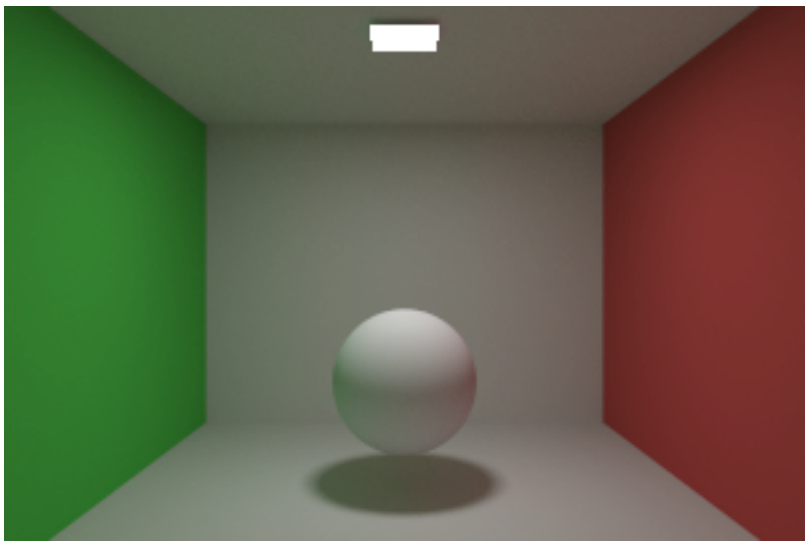
Figure 6.3: Image reconstruction comparison from 25% samples



(c) Reconstructed from quasi-random sampling



(d) Reconstructed from regular sampling



(e) No reconstruction

Figure 6.3: Image reconstruction comparison from 25% samples

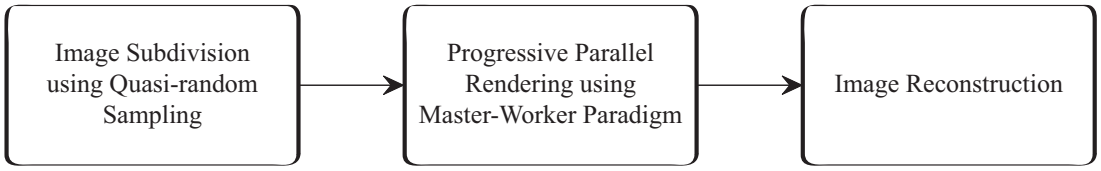


Figure 6.4: The pipeline for the straightforward approach.

6.2.1 Straightforward Approach

For rendering high-fidelity images, several computations are needed per pixel. As explained earlier in Section 6.1, these per pixel computations can be carried out in parallel, independently of one another. The idea behind this algorithm is to exploit this inherent parallelism of the rendering computations by utilising changeable resources. The pipeline for this algorithm is presented in Figure 6.4.

6.2.1.1 Task Subdivision and Fault-tolerance

While rendering in parallel, an image is traditionally divided into groups of neighbouring pixels which are computed independently on parallel processors. The disadvantage of using such an approach while computing on variable resources such as a desktop grid is that if a task fails, image reconstruction techniques would not be able to approximate the missing data as there would be significant gaps in the image. Duplicate tasks would need to be scheduled in order to cope with faults, as is done in most applications which provide fault-tolerance using redundancy. However, the presented algorithms avoid such redundancy by subdividing the image into sets of pixels using a quasi-random sampling strategy (see Figure 6.1) instead of using tiling. These groups of pixels are scheduled as different tasks which are completed in parallel. A well known image reconstruction technique, nearest-neighbour reconstruction (see Section 2.7.1), is then used to fill in the missing data. A single buffer is stored at the master to accumulate the results and if some pixels are missing then this buffer is reconstructed to obtain the final result.

6.2.1.2 Task Scheduling and Load Balancing

A master-worker paradigm (see Section 3.5) is used for rendering in parallel. All the tasks to be completed are kept in a queue on the master. This is a pull-model where each idle worker sends a request to ask for work from the

Figure 6.5: Image subdivision based on components.

master. The master then assigns it a task from the front of the task queue. Using fine granularity, this demand-driven dynamic task scheduling maintains a well-balanced load.

6.2.1.3 Time-constraint

The master keeps track of the time and whenever it assigns a task to the worker for the first time, it sends it a timestamp for the deadline. The worker checks if this timestamp is about to expire, while rendering the image. At a small Δt time before the timestamp is about to expire, it sends back the data for all the pixels for which the computations have been completed. The value of Δt is chosen such that the results from the worker would reach the master before the time-constraint expires. The master discards any results received after the deadline and stops scheduling any new tasks. It then uses reconstruction methods, if need be, to generate the final image. The reconstruction is done in real-time using a commodity GPU.

6.2.2 Component-based Algorithm

The motivation for this algorithm is to be able to achieve better visual quality by limiting the reconstruction noise. The division of the computations at a pixel level allows a finer granularity meaning more pixels can be scheduled per job. Therefore, with each completed job in a given time-constraint more information is obtained, reducing the dependency on image reconstruction techniques. The lighting calculations at a pixel level are subdivided into components.

The radiance at a point x in direction Θ is given by the rendering equation (Equation(2.5)):

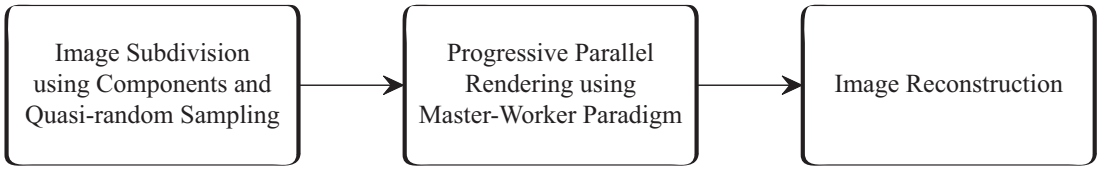


Figure 6.6: The pipeline for the component-based approach.

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Theta \leftrightarrow \Psi) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$

For a specific component i,

$$L_i(x \to \Theta) = \int_{\Omega_x} f_r(x, \Theta \leftrightarrow \Psi_i) \cos(N_x, \Psi_i) L_i(x \leftarrow \Psi_i) d\omega_\Psi$$

Therefore,

$$L(x \to \Theta) = L_e(x \to \Theta) + \sum L_i(x \to \Theta)$$

Traditionally, it is common to subdivide the computation into direct(L_d) and indirect(L_{id}) computations [SSH*98, DSSC05]:

$$L(x \to \Theta) = L_e(x \to \Theta) + L_d(x \to \Theta) + L_{id}(x \to \Theta)$$

Furthermore, the indirect computations can be broken into separate components such as indirect diffuse, indirect specular and indirect glossy. The component-based approach divides the light components into indirect diffuse which shall be referred to as indirect lighting and all the other non-indirect diffuse components are grouped together which shall be referred to as direct lighting as shown in Figure 6.5. The pipeline for component-based approach is presented in Figure 6.6.

6.2.2.1 Component-based Task Subdivision

Task subdivision in the straightforward approach is further enhanced by the component-based approach, by subdividing computations into components on a per pixel level as well. This is in addition to image space subdivision employed in the straightforward approach as explained in Section 6.2.1.1. The lighting calculations are primarily broken into two components, namely direct and indirect

(a) Set of 16 images depict the approximation of indirect lighting. Each of these images has been generated by sampling a different set of 16 VPLs.

(b) Composite of all the 16 images representing the indirect lighting estimation. ($\alpha = 256$, $\beta = 16$)

Figure 6.7: Indirect lighting calculation for the Sibenik model

lighting calculations. The direct light is approximated by ray tracing the scene using different samples on area light sources. This computation can be split such that a group of tasks generating the complete image, samples a different point on the area light source. The direct lighting can then be obtained by averaging results from these tasks. The estimation of indirect lighting is done by sampling Virtual Point Lights (VPLs) (see Section 2.6.4). Light paths are traced by shooting rays from the light source and the VPLs are created at the points where these intersect with the scene. These VPLs are then sampled with visibility rays to obtain the indirect lighting. If α VPLs are to be sampled for generating the image, they can be grouped into β sets where each set contains α / β VPLs. Each group of tasks generating the whole image can then sample one of the β subsets independently of the other groups [KH01]. The images generated by the different groups of tasks, then need to be averaged to obtain the complete estimation of the indirect lighting calculation.

6.2.2.2 Fault-tolerance

The fault-tolerance mechanism used is the same as explained in Section 6.2.1.1, wherein reconstruction techniques are used for dealing with faults. A single buffer is stored at the master to accumulate the direct light samples and if some pixels are missing then this buffer is reconstructed. Multiple buffers are stored for indirect light samples, one for each set of VPLs being sampled together. Each set of VPLs generates a considerably different image than another set as shown in Figure 6.7. Therefore, to be able to reconstruct the indirect lighting with a better visual fidelity, multiple buffers are used for each set of VPLs. Furthermore, an indirect buffer is discarded if it does not contain a minimum percentage of data (chosen to be 25%) and there are other buffers which contain more than the minimum amount. This is done to prevent the reconstruction noise from having a big impact on the visual quality of the image.

6.2.2.3 Task Scheduling and Load Balancing

A similar approach is used for task scheduling and load balancing as explained in Section 6.2.1.2. Each task computes a part of one of the lighting components (direct or indirect) for a group of quasi-randomly chosen pixels. Two task queues are maintained at the master, one contains the tasks for direct lighting computations and the other for indirect lighting computations. Tasks are chosen alternately

Scene	Time-constraint (in seconds)	Component-based Approach				Straightforward Approach	
		Number of Tasks Completed		Percentage of Reconstructed Pixels		Number of tasks completed	Percentage of Reconstructed Pixels
		Direct Light	Indirect Light	Direct Light	Indirect Light		
Conference Room	10.0	727	726	0.00	29.10	1586	22.6
	8.0	515	514	0.00	0.00	991	51.66
	4.0	214	214	0.00	52.45	479	76.66
Sibenik	10.0	653	653	0.00	36.34	1440	29.75
	5.0	154	154	0.00	52.34	663	67.68
	3.0	95	94	0.00	51.66	298	85.5
Cornell Box	6.0	377	377	0.00	50.91	1125	45.12
	4.0	124	123	0.00	51.95	565	72.46
	2.0	21	21	67.18	67.23	130	93.7

Table 6.1: Component-based approach versus straightforward approach

from the two queues when the workers send a request for more work. A set of tasks computing the direct lighting for different groups of pixels by sampling the same point on the area light sources are queued together. A group of tasks which computes the indirect lighting from a given set of VPLs is scheduled in two equal parts. The first part of the group which calculates fifty percent of the pixels is scheduled before another group of tasks which calculates the complementary half of the pixels using another set of VPLs [KH01]. The second part of the group which uses the first set of VPLs is scheduled after one half of all the tasks have been scheduled. This is done so as to give a better visual quality by sampling more sets of VPLs within a given time-constraint, as the indirect lighting can be reconstructed if the time-constraint does not permit the completion of all tasks.

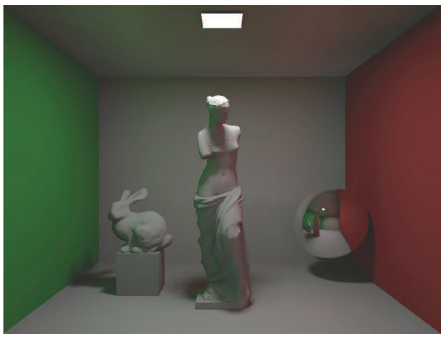
The same approach as mentioned in Section 6.2.1.3 is used for handling the time-constraints.

6.3 Results

The presented algorithms have been implemented using the Condor Master-Worker framework [GKYL01] and run on a desktop grid formed by connecting twenty four machines with two dual-core AMD Opteron processors running at 2.6GHz at each node. Each machine also had 8GB RAM shared among the four CPU cores. Each core was used independently, as the number of idle CPUs in a machine vary with the load on it. Such a testbed was chosen because it made

(a) Sibenik (80k)

(b) Conference Room (190k)



(c) Cornell Box (63k)

Figure 6.8: The scenes used for the experiments. The number in brackets indicates the polygon count of the model.

it possible to compare the visual quality of the two proposed algorithms. Experiments were conducted for time-constraints and fault-tolerance and the results obtained are detailed in the following section.

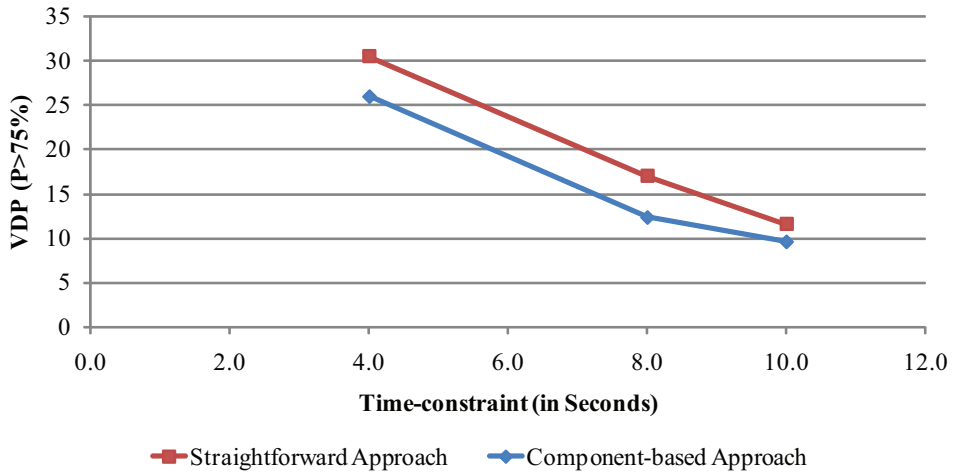
6.3.1 Time-constraints

The presented algorithms have been compared and the results showing the progression of the computation at different time-constraints are shown in Table 6.1. Figure 6.8 shows the three scenes that were chosen for comparison. The image resolution used was 1024×768 and 16 samples per pixel were used for direct lighting while the indirect lighting was calculated using 256 VPLs. The image was broken into 64 groups of pixels for the component-based approach. Each of these groups computed a single sample for the direct lighting and hence 1024 tasks were used in total for computing 16 samples per pixel. For the indirect lighting computations, 256 VPLs were split into 16 sets of 16 VPLs each. Hence 16 buffers were used at the master for storing the indirect computations. Here, also the

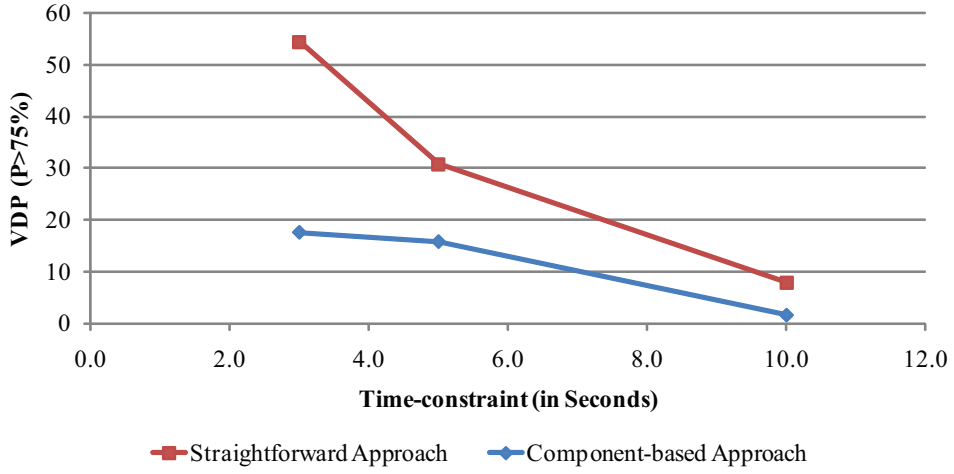
image was subdivided into 64 groups of pixels and a total of 1024 tasks was used to calculate the indirect lighting. For the straightforward approach the whole image was subdivided into 2048 groups of pixels so that the number of tasks to be completed for the whole solution remained the same for both approaches. For each of these 2048 groups, the task computed 16 samples per pixel for the direct lighting and sampled all 256 VPLs for the indirect lighting.

The visual quality of the images generated using the two approaches was compared using a well-known Visual Difference Predictor (VDP) metric [Dal93]. This metric is used for predicting the probability with which two images would be perceived differently by a human. The VDP (P> 75%) denotes the percentage of pixels in an image that would be perceived differently with a probability higher than 75%. The images rendered without any time-constraint were used as the gold standard while using the VDP for error prediction. It was found that the time needed for generating the gold standard images for both approaches was almost the same. The VDP results are shown in Figure 6.9. It can be seen from Figure 6.9, that the error in visual quality as predicted by VDP for a given time-constraint has been found to be always less for the images computed using component-based approach as compared to the straightforward approach. The difference between the two approaches is considerably higher for a shorter time-constraint. This variation is more evident for the Sibenik Model than the Cornell Box because the impact of reconstruction noise on visual quality is higher for a complex scene. The visual quality of the two approaches appears to converge as the time-constraint is increased. Figure 6.10, depicts the images of Sibenik model at a time-constraint of 5 seconds for both approaches along with the VDP output images.

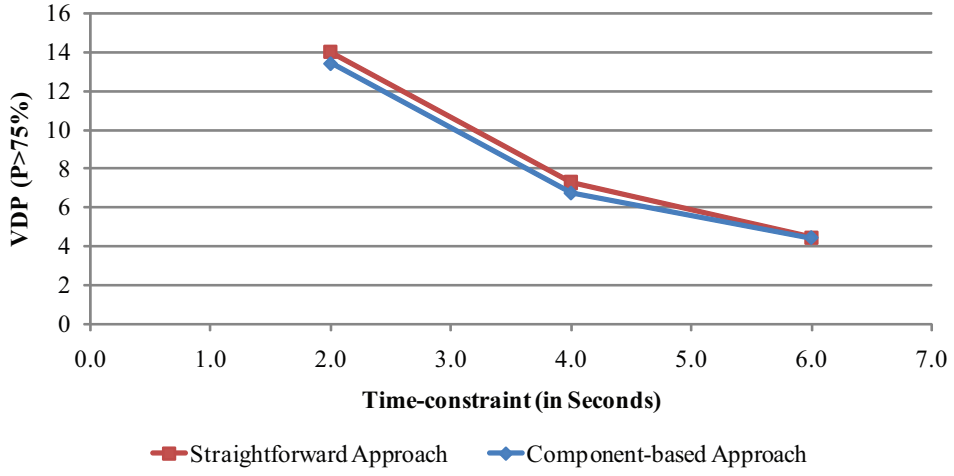
An important thing to note here is that, as advised by Ramanarayanan et al. [RFWB07], VDP is a useful metric but not a substitute for the human eye. To illustrate this fact, parts of Cornell Box images generated using the two presented approaches with a time-constraint of two seconds are shown in Figure 6.11. The values of VDP(P> 75%) obtained are 13.93% and 13.43% (as shown in Figure 6.9) whereas the image generated by the component-based approach is much more visually appealing than the one generated by the straightforward approach as it has lower reconstruction noise.



(a) Conference Room



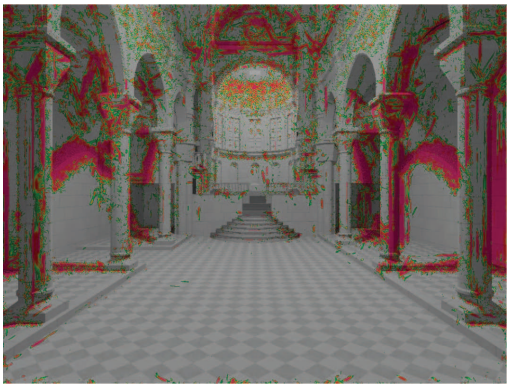
(b) Sibenik



(c) Cornell Box

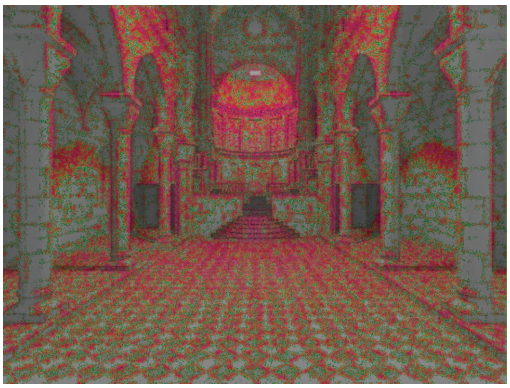
Figure 6.9: The VDP results.

(a) Component-based approach



(b) VDP for component-based approach

(c) Straightforward approach

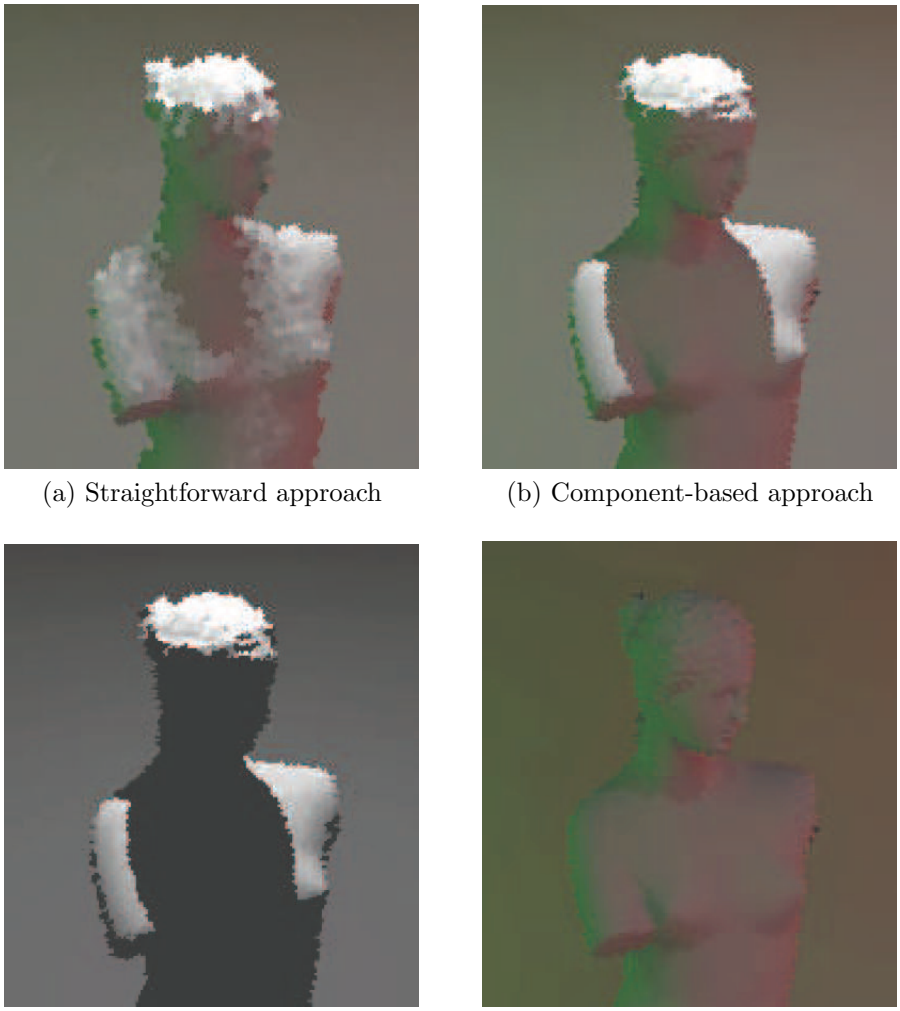


(d) VDP for straightforward approach

Figure 6.10: VDP comparison for images of the Sibenik model generated with a time-constraint of 5 seconds. In the VDP output, the grey pixels depict no perceivable difference. The green pixels are used to represent low probability of noticeable difference, while the red pixels are used for depicting high probability.

6.3.2 Fault-tolerance

In order to illustrate the fact that the presented algorithms are capable of producing results on volatile resources handling both addition and removal of resources, the resources were varied in a controlled fashion. In one run, the resources were increased from 14 machines (56 cores) to 24 machines (96 cores) linearly with one machine being added each second and the time constraint used was 10 seconds. The VDP (P> 75%) was 11.15% when compared to the gold standard. In the second run, the resources were decreased from 24 machines (96 cores) to 14 machines (56 cores) linearly with one machine being removed each second and the time-constraint used was again 10 seconds. The VDP (P> 75%) was found to be 11.31% when compared to the gold standard. This is close to VDP (P> 75%)



(a) Straightforward approach

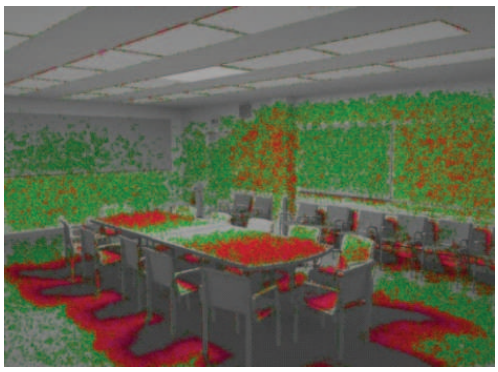
(b) Component-based approach

(c) Direct lighting for component-based approach

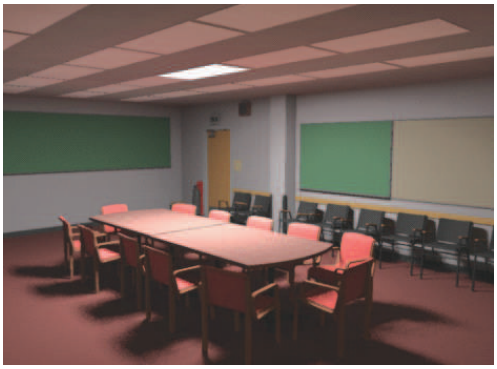
(d) Indirect lighting for component-based approach

Figure 6.11: A part of Cornell Box image is shown to compare visual quality for the component-based approach with the straightforward approach at a time-constraint of 2 seconds.

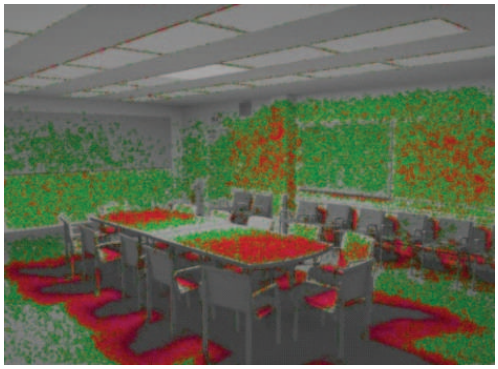
(a) Increasing resources



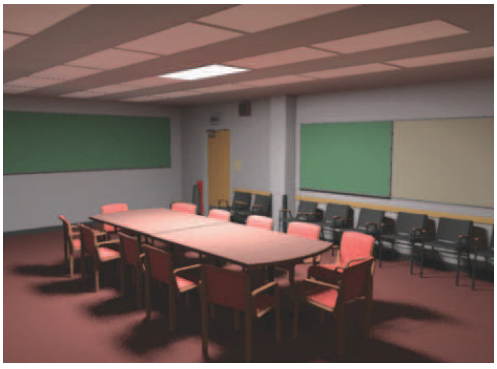
(b) Increasing resources - VDP



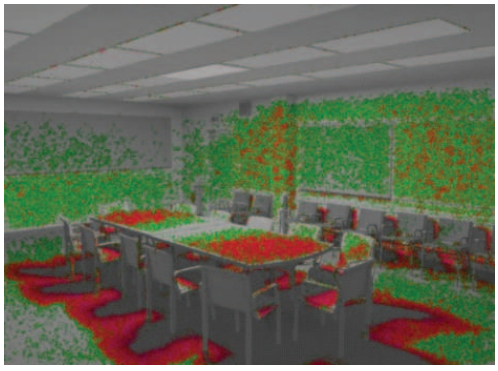
(c) Decreasing resources



(d) Decreasing resources - VDP



(e) Constant resources



(f) Constant resources - VDP

Figure 6.12: Fault-tolerant nature of the component-based algorithm.

value of 9.55% for the image of the Conference Room generated without varying the resources with a time-constraint of 10 seconds. The component-based algorithm was used for these results as it provides better visual fidelity. The resulting images are shown in Figure 6.12.

6.4 Summary

This chapter presented two novel algorithms for rendering in parallel on volatile resources within a user-defined time-constraint. It demonstrates the use of desktop grids for computing high-fidelity images. Furthermore, it illustrates that time-constraints are a good way of using resources which are dynamic in nature. Fault-tolerance is also handled without using redundancy, especially under time-constraints.

A comparison between the two presented approaches has been carried out and it has been shown that the component-based approach offers a better visual quality over the straightforward approach. This can be attributed to the fact that the component-based approach increments the visual quality in steps and is less dependent on image reconstruction techniques when compared to the straightforward approach.These algorithms can be further extended to render high-fidelity animations on desktop grids as well (see Chapter 8), where computations for each frame of the animation can be parallelised using the presented approaches and tasks from all the frames can be queued up on the master. Although care must be taken to filter reconstruction noise between frames to prevent flickering.

It has been observed that the current frameworks for task management on desktop grids are developed with an aim of trying to provide guarantee of service for the applications on volatile resources by using traditional fault-tolerant mechanisms (see Section 3.4.1), which hinder performance. The ramp-up time of Condor master-worker framework was also found to be substantial requiring up to a minute to start the computations. These issues will make it difficult to achieve interactive rates which requires high performance computing and it is one of the final goals for high-fidelity rendering. Off-line rendering on the other hand, can cope with faults without relying on such fault-tolerance techniques as explained in Section 6.2.1.1. Hence, a task management framework with dynamic task scheduling which will focus on achieving maximum throughput will enhance the performance of desktop grids even further from a rendering perspective, as

discussed in the next chapter.

CHAPTER 7

Interactive Rendering on Desktop Grids

High-fidelity interactive rendering has been traditionally restricted, and interactive rendering algorithms are currently incapable of seamlessly handling the variable computational power offered by the non-dedicated resources of a desktop grid. This chapter presents a novel fault-tolerant algorithm for rendering high-fidelity images at interactive rates which is capable of handling variable resources. A conventional approach of rescheduling failed jobs in a volatile environment would inhibit performance while rendering at interactive rates as the time margins are small. Instead, the proposed method uses quasi-random sampling along with image reconstruction techniques to deal with faults in a similar way to the algorithms described in the previous chapter. This enables users to experience interactive high-fidelity rendering on their desktop machines.

This chapter originally published as [ADBR*10] is organised as follows: Section 7.1 introduces the chapter. Section 7.2 discusses the novel interactive parallel rendering system. Its implementation is described in Section 7.3. The evaluation of the system is presented in Section 7.4 and finally the chapter is summarised in Section 7.5.

7.1 Introduction

High-fidelity rendering at interactive rates is essential for providing an immediate feedback to the user for steering any visualisation towards a better solution. In the context of rendering computer generated imagery, especially for areas such as product visualisation, the film industry or an architectural walk-through, it would help an animator to adjust lighting conditions and object properties in the scene for the best visual appeal. However, such interactive renderings require

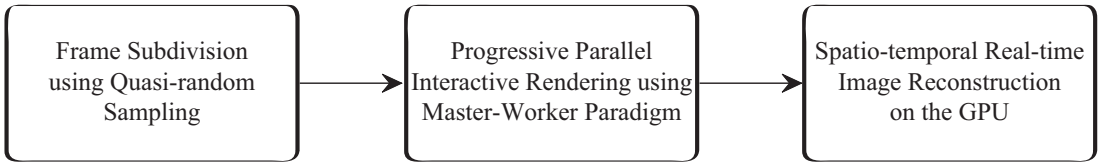


Figure 7.1: The pipeline for interactive rendering on a desktop grid.

large amount of computational power and thus are traditionally rendered using dedicated parallel resources, often referred to as render farms. These dedicated render farms are expensive, limiting the number of users who have access to high-fidelity interactive rendering. In addition, only a few implementations for high-fidelity interactive rendering exist even on these dedicated machines, for example Benthin et al. [BWS03] and Wald et al. [WKB*02] (see Section 2.9). The method proposed in this chapter effectively enables the majority of users that work in small to medium sized companies or universities to experience interactive high-fidelity rendering on their desktop machines when other users' resources are left idling, without the expensive requirements of a dedicated render farm.

Interactive rendering imposes severe time-constraints on the system and as mentioned before in Section 6.1, traditional fault-tolerance mechanisms (see Section 3.4.1) are not well-suited under such constraints as they inhibit performance. Instead robust image space sampling techniques such as quasi-random sampling along with image reconstruction methods can be used to deal with faults (see Figure 6.2). If a job fails, image reconstruction algorithms can be used to fill in the missing data. The approach presented in the previous chapter relied on grid middleware for job management and scheduling which is not designed for interactivity. Furthermore, a spatial nearest neighbour reconstruction as employed earlier would not be sufficient, as shown in Section 7.4.3. Hence, this chapter presents an enhanced scheduling strategy which adjusts the computations by monitoring the variation in computational power and uses advanced spatio-temporal reconstruction for a better visual fidelity. The pipeline for this approach is shown in Figure 7.1.

The idea of combining sparse sampling with reconstruction has been used before (see Section 2.7.2) for ray tracing at interactive frame rates. The presented approach extends this idea by adapting sparse sampling as a mechanism to enable fault-tolerance. When exploited in the context of interactive rendering of high-fidelity images, it allows the user to change the scene at run-time and receive

feedback without employing expensive dedicated resources. This is interesting for any application that can benefit from interactive hypothesis testing and in which creativity may be stifled by the need to wait for significant periods of time before seeing any rendering results from a small change to a model. A user may not be able to try out all the different settings in an offline rendering environment as it is a time consuming process and could use this system instead without any additional expense by just connecting all the available machines into a desktop grid.

Previous algorithms for offline rendering on a computational grid [ACD08, CSL06] relied on the fault-tolerance provided by the grid middleware to tackle any faults. A time-critical visualisation method for grid computing presented by Gao et al. [GLH*08] used redundancy for fault-tolerance. The time-constrained algorithms for image rendering on a desktop grid presented in the previous chapter, used quasi-random sampling with reconstruction to cope with faults. But that approach could only handle static images due to the dependence on the grid middleware for job management and could not achieve interactivity. The method presented in this chapter describes a novel system that enables interactive rates, which would not have been possible by relying on traditional grid middlewares for job management and fault-tolerance.

7.2 Fault-tolerant Interactive Rendering System

In this section, the design and functioning of the proposed method are described. It follows a master-worker paradigm using a job pull mechanism (see Section 3.5). These workers are scheduled on the available resources of a desktop grid and they connect to the master to fetch a rendering job. Workers can connect and disconnect at their own will and the master has no control over this. This enables the method to employ variable resources for parallel rendering. Each frame which needs to be rendered is divided into quasi-random sets of pixels and each set is queued up as a different job by the master. The number of jobs per frame is much greater than the maximum number of available resources for load balancing. If some of these jobs are not completed, reconstruction is used for completing the frame.

The overview of the approach is presented in Figure 7.2. First, the user interacts with the scene which defines the rendering parameters for the frame. This

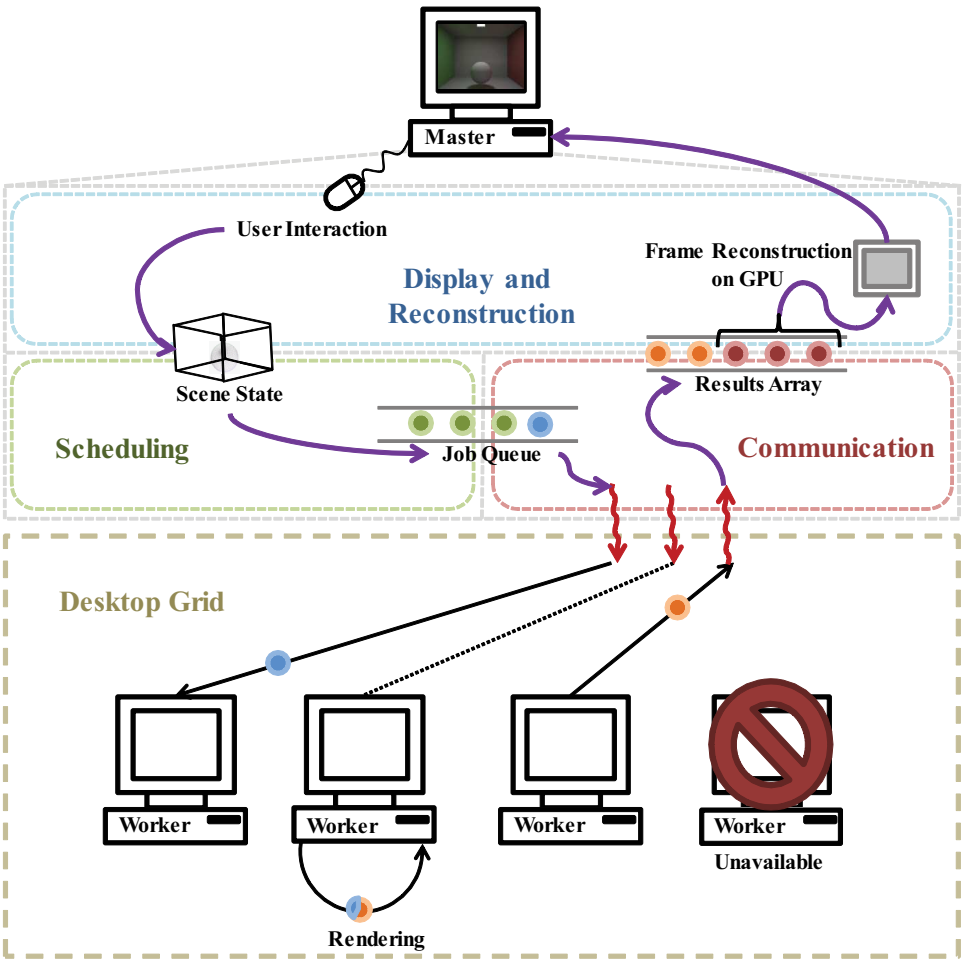


Figure 7.2: The overview of the interactive rendering system showing the interactions inside the master and between the master and the desktop grid.

frame is then divided into sets of quasi-randomly chosen pixels which are queued up as jobs. When an idle worker connects to the system, a job from the queue is sent to it to process. Once the results are received, the partial frame is sent to the GPU for reconstruction and display. Figure 7.2 also illustrates the four possible states of any machine in the desktop grid: receiving a job, rendering, sending results and unavailable for computation. The following subsections provide the details about communication, scheduling, display and reconstruction, and workers used in the presented method.

7.2.1 Communication

The master uses one thread per worker architecture for communicating with them for maximum throughput. These threads have been designed such that they do not need to synchronise with each other. Also, they use non-blocking data structures while synchronising with the scheduler and display and reconstruction threads allowing the master to scale efficiently with the increase in number of workers.

The master sends out a job packet from the front of the job queue and then waits for the worker to process the job. When the worker sends the results back, these are stored for later compositing. The job packet consists of the following:

Frame Number identifies the frame to which this job belongs.

Set Number is used by the worker to identify which set of quasi-random pixels it needs to render.

Scene State includes all the information which the worker needs to render the correct view of the scene such as camera position, object materials, object positions etc.

The job packet contains the absolute scene state rather than having an update scene message, because a worker may or may not receive all the update messages depending on which frames it processes. This helps in synchronising the scene state on all the workers processing a given frame. The result packet contains the luminance values of the computed pixels along with the frame number and the set number according to which job was processed by the worker. As multiple frames may be scheduled together, the master can receive results for multiple frames at the same time. Therefore, an array of results is used, and the frame and set numbers correctly identify the results. It is possible that a worker takes too long a time to send back the results for a frame which, by then has been displayed. In such a case these results are discarded and a new job is sent to the worker.

7.2.2 Scheduling

The computing power of a desktop grid is time-variant and hence, it needs to be closely observed to prevent either an excess or lack of jobs in the queue. The job scheduler monitors the number of jobs in the job queue. If this number falls

below the job queue threshold, it adds jobs for the current frame, using the latest scene state, to the job queue. The scheduler works asynchronously to match the rate at which jobs are added to the rate at which they are being processed by the workers in a desktop grid to avoid under- or overflow of the queue.

7.2.3 Display and Reconstruction

The interactive system is fully dynamic such that it allows the user to change camera view, object positions, object materials, lighting positions and lighting conditions while interacting with the scene. This interaction sets the scene state which the scheduler uses for queuing up the jobs. Before displaying a frame, the master needs to wait for the results to come back from the workers. It then reconstructs the partial frame on the GPU as explained further in Section 7.2.3.1, and displays it on the user's screen. It is possible that the job queue contains jobs for a frame which has been already displayed and in such a case those jobs are discarded from the job queue.

A heuristic is used to guide the master to determine how long to wait for the results from the worker before reconstructing the partial frame. There are two parameters which form the heuristic, a quality constraint and an interaction constraint.

The quality constraint Q_t, at a time instant t measured from the start of computation for the current frame, is given by:

$$Q_t = \frac{\text{number of jobs finished for the current frame}}{\text{number of total jobs for the current frame}}$$

The quality constraint controls the visual quality of the reconstructed frame. The interaction constraint I_t, at a time instant t is given by:

$$I_t = \frac{\text{time taken for the current frame}}{\text{time taken for the last frame}}$$

The interaction constraint estimates the wait time using the previous frame and tries to maintain smooth user interaction even with variable computing resources. The master waits for time t while one of the following conditions is true:

$$\begin{aligned} Q_t < Q_{max} \quad &\wedge \quad I_t < I_{min} \\ Q_t < Q_{min} \quad &\wedge \quad I_t < I_{max} \end{aligned}$$

where Q_{min}, Q_{max}, I_{min} and I_{max} are user-defined minimum and maximum values for the quality and interaction constraints respectively. In the worst case scenario,

if the master does not receive results to meet the minimum quality constraint in the time specified by maximum interaction constraint, it skips the current frame and moves on to the next frame. This ensures that even if most of the workers computing the same frame go down, the master would be able to adjust to this significant transient variation in the compute power.

If the user requires a fixed frame rate, the master waits for the specified amount of time before moving onto reconstruction. In this case, the reconstruction tackles the variability of resources and the visual quality of the frames changes.

7.2.3.1 Image Reconstruction

While rendering using volatile resources, some results may not be received due to faults. Therefore, image reconstruction techniques need to be employed to fill in the missing pixels in a partial frame. A discontinuity function, $D_{i,j}$ is used to find if two pixels i and j are similar for interpolating across them. The discontinuity function is given by:

$$D_{i,j} = \max(0, \vec{n_i} \cdot \vec{n_j} - \alpha) \times \max(0, |z_i - z_j|^2 - \beta^2)$$

where,

$\vec{n_i}$ = Orientation at pixel i

α = User-defined orientation threshold

z_i = Position at pixel i

β = User-defined position threshold

This discontinuity function can be used as a spatial discontinuity filter when i and j are different in spatial positions, and it can also be used as a temporal discontinuity filter when i and j represent same pixel in different frames.

Firstly, a nearest neighbour reconstruction technique is used to estimate the missing pixels. This technique is chosen over other reconstruction methods due to its simplicity and the ability to run in real-time on a GPU. The luminance L_i, of a missing pixel i is calculated as:

$$L_i = \frac{\sum_{j \in \mathbb{N}\{i\}} L_j \times D_{i,j} \times k_{i,j}}{\sum_{j \in \mathbb{N}\{i\}} D_{i,j} \times k_{i,j}} \qquad \forall i \in \Omega$$

where,

L_i = Luminance at pixel i

$\mathbb{N}\{i\}$ = Set of calculated pixels in the neighbourhood of i

$D_{i,j}$ = Spatial discontinuity function

$k_{i,j}$ = User-defined weighting function

Ω = Set of all missing pixels

After nearest neighbour reconstruction, luminance values are accumulated and displayed if the scene state is unchanged from the previous frame. If this is not the case, then a temporal filter is applied to reduce flickering when user interaction occurs. The temporal discontinuity function removes the ghosting artefacts usually generated by a temporal filter and the luminance is updated as follows:

$$L_i = \frac{\sum_{j \in \mathbb{T}\{i\}} L_j \times D_{i,j} \times w_{i,j}}{\sum_{j \in \mathbb{T}\{i\}} D_{i,j} \times w_{i,j}} \qquad \forall i \in \Pi$$

where,

$\mathbb{T}\{i\}$ = Set containing previous values of pixel i

$D_{i,j}$ = Temporal discontinuity function

$w_{i,j}$ = User-defined weighting function

Π = Set of all image pixels

Finally, the spatial noise due to low samples per pixel, while the user is interacting with the scene, is reduced by applying a Gaussian filter with spatial discontinuities . This filter is given by:

$$L_i = \frac{\sum_{j \in \mathbb{M}\{i\}} L_j \times D_{i,j} \times g_{i,j}}{\sum_{j \in \mathbb{M}\{i\}} D_{i,j} \times g_{i,j}} \qquad \forall i \in \Pi$$

where,

$$g_{i,j} = \frac{e^{-\frac{i^2+j^2}{2\sigma^2}}}{2\pi\sigma^2}$$

σ = Standard deviation of the Gaussian function

$\mathbb{M}\{i\}$ = Set of pixels in the neighbourhood of i

This reconstruction and filtering is only done on the luminance values calculated by the renderer, which does not contain the information about the albedo (surface colour of the material at the primary ray intersection that does not take into account any lighting). The albedo is calculated on the GPU and multiplied afterwards. This prevents filtering across colour boundaries contained in the albedo such as high frequency texture details.

7.2.4 The Worker

The worker is scheduled on any machine that becomes available on the desktop grid at any time. It is implemented as a single thread such that any core on a machine can be employed, if it lies idle. The worker connects to the master asking for work and receives the job packet with a description of the rendering job. It changes the scene state according to the description provided in the job packet and then renders the specified set of quasi-random pixels. It calculates the luminance of those pixels and sends them back to the master in a result packet and waits until the master sends more work in the next job packet.

7.3 Implementation Details

The interactive system described in the previous section was implemented and tested on a desktop grid. The TCP/IP protocol was used along with portable data serialisation to communicate between workers running on heterogeneous machines and the master. Condor (see Section 3.6.2) was employed for managing workers on vacant resources and transferring the initial scene files and executables to start computation on them. The test platform was a desktop grid consisting of two kinds of processors connected by a 100 Mbps Ethernet LAN. There were 48 Dual Core 2.6 GHz AMD Opteron processors with 4GB RAM each running Linux and 8 Quad Core 3.0 GHz Intel Extreme processors with 4GB RAM running Windows. Each of the 128 cores was treated as a separate resource to obtain finer granularity. The master ran on a machine having 2 Dual Core 2.6 GHz AMD Opteron processors with a shared memory of 8GB running Linux. The machine also had an NVidia 8800 GTX GPU for reconstructing the partial frames in real-time using the OpenGL shading language. The reconstruction was done on the GPU for optimum performance. The path tracing algorithm (see Section 2.6.2) was used for calculating the luminance values of the pixels on the

workers, however, other point sampling methods could also be employed. Each job consisted of rendering a set of pixels chosen quasi-randomly using Sobol and van der Corput sequences (see Chapter 4) from a 640×480 frame with four samples computed per pixel per update. The choice of four samples per pixel was made to achieve interactivity on the test bed. Since it has a direct impact on the amount of computation required, much more computational power would be needed to compute higher number of samples per update and maintain interactive frame rates. As desktop machines gradually improve, future iterations of this system would use more samples per pixel. Furthermore, to mask the effect of such low sampling, the filtering was applied as described in Section 7.2.3.1. Once the user stopped at a particular scene state, such filtering was removed and samples were accumulated to offer a refined view. The method allowed the user to interactively change an object's position, material or colour along with a change of light position or direction or change the environment map in a scene while viewing it from different positions.

The master used $Q_{min} = 20\%$, $Q_{max} = 50\%$, $I_{min} = 2$ and $I_{max} = 4$ for estimating the wait time using the heuristics described in Section 7.2.3. The job queue threshold was twice the number of jobs per frame. A radius of 5 pixels was chosen, and $k_{i,j} = 1$ to give equal weights for the nearest neighbour reconstruction step. For the temporal filter, data from the four previous frames was used. A weighting function $w_{i,j}$, provided a lesser contribution from older pixel values:

$$w_{i,j} = \frac{1}{F_i - F_j}$$

where, F_i represents the current frame number and F_j represents the older frames. The Gaussian filter kernel was seven pixels wide with $\sigma = 1$, while $\alpha = 0.75$ and $\beta = 1$ was used for the discontinuity function. These values were chosen so that the reconstruction could be performed in real-time on the GPU.

7.4 Evaluation

Using the desktop grid described in the previous section, interactivity was achieved for a variety of scenes. Even for a complex model such as Kalabsha (see Figure 7.3b) with 861k polygons, large indirectly lit areas and two light sources, the sky and a directional light, the system achieved about 3 frames per second on the test platform. For a simpler scene such as the Kiti model (243k polygons), a fixed frame rate of up to 10 frames per second could be obtained.

(a) Cornell Box (63k)

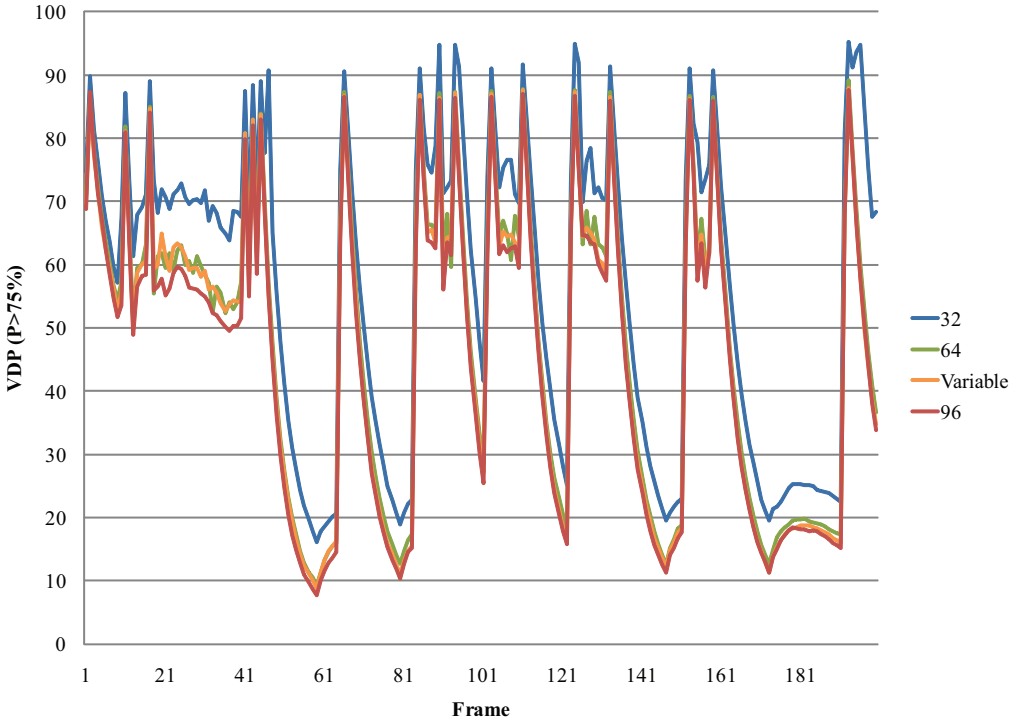
(b) Kalabsha (861k)

(c) Kiti (243k)

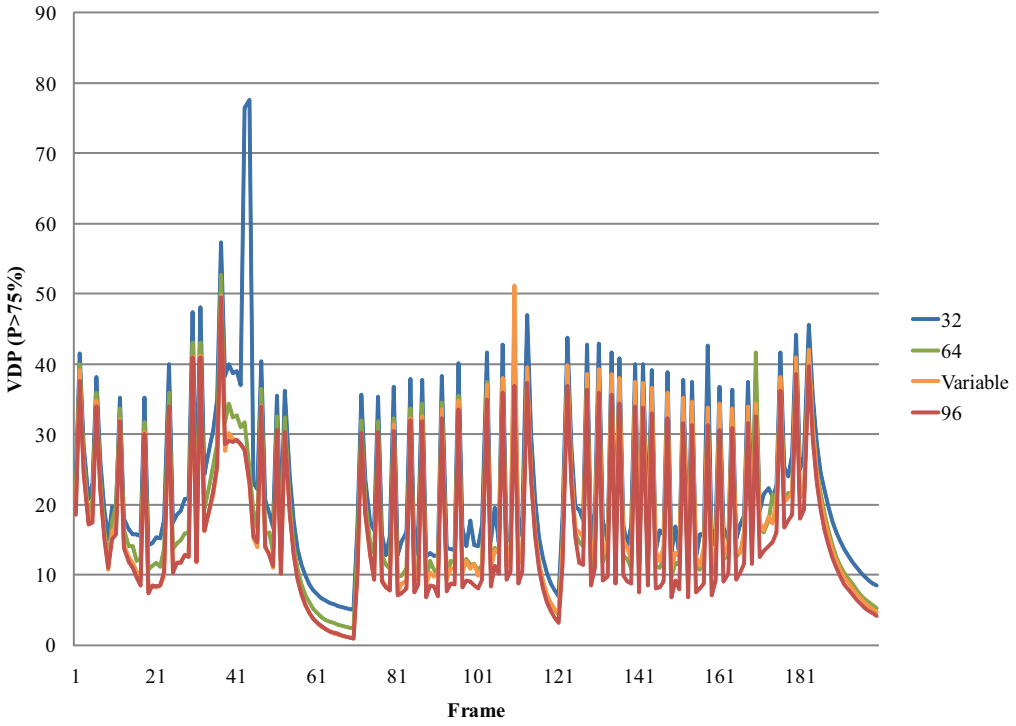
(d) Race Car (69k)

(e) Sponza (66k)

Figure 7.3: The scenes used for evaluating the interactive system. The number in brackets indicates the polygon count of the model.

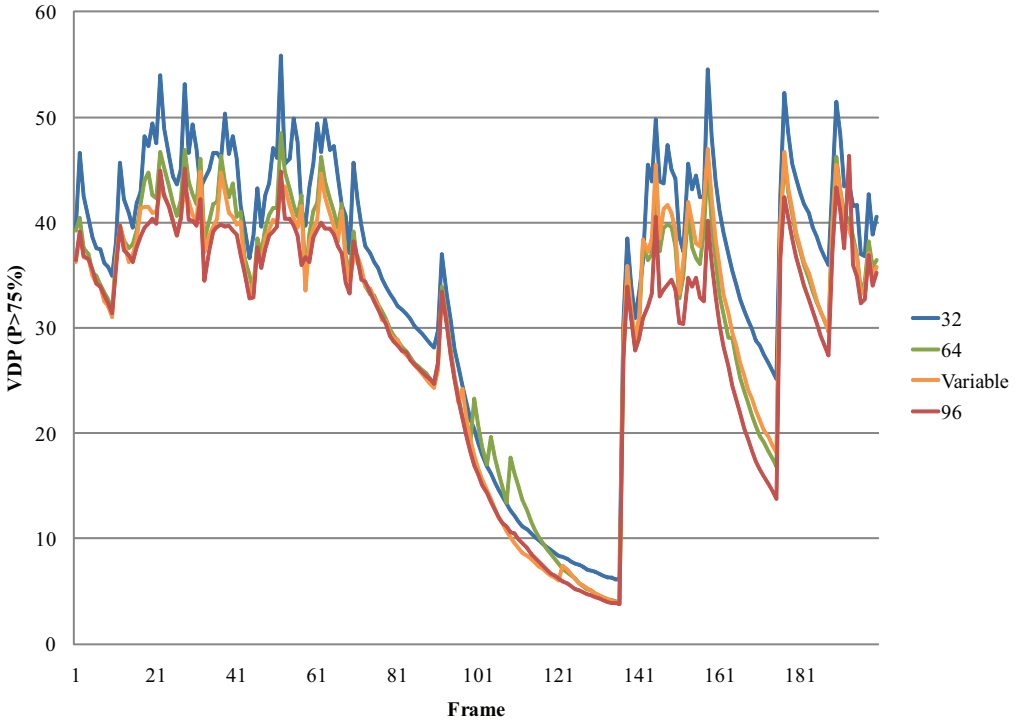


(a) Cornell Box

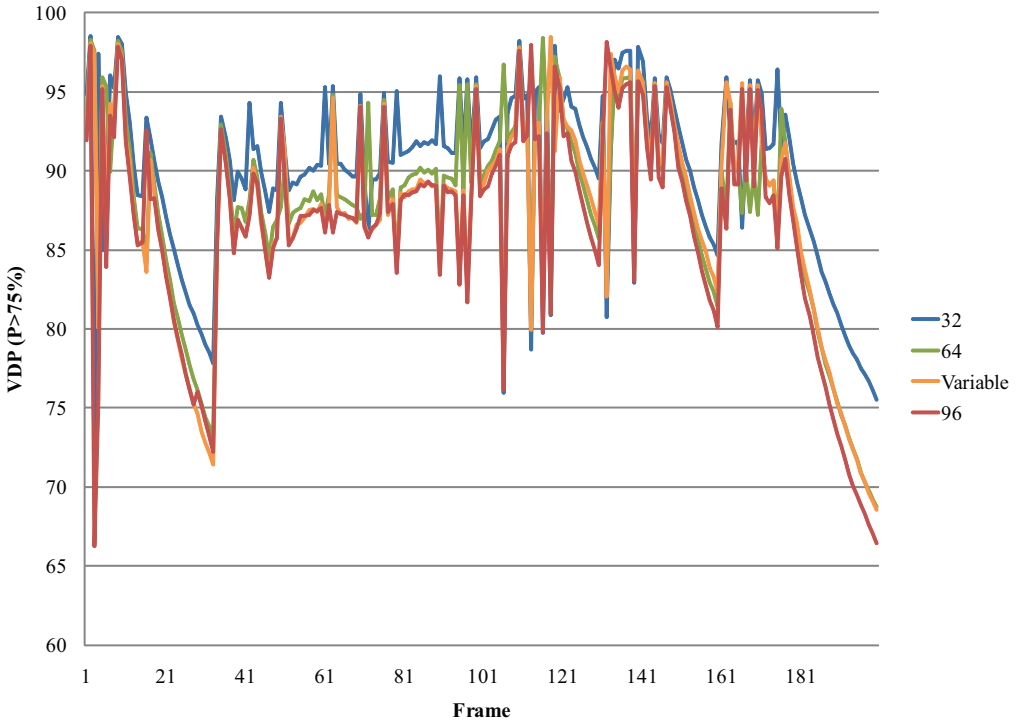


(b) Kiti

Figure 7.4: The comparison of visual quality of an interactive sequence with different resources for four scenes.

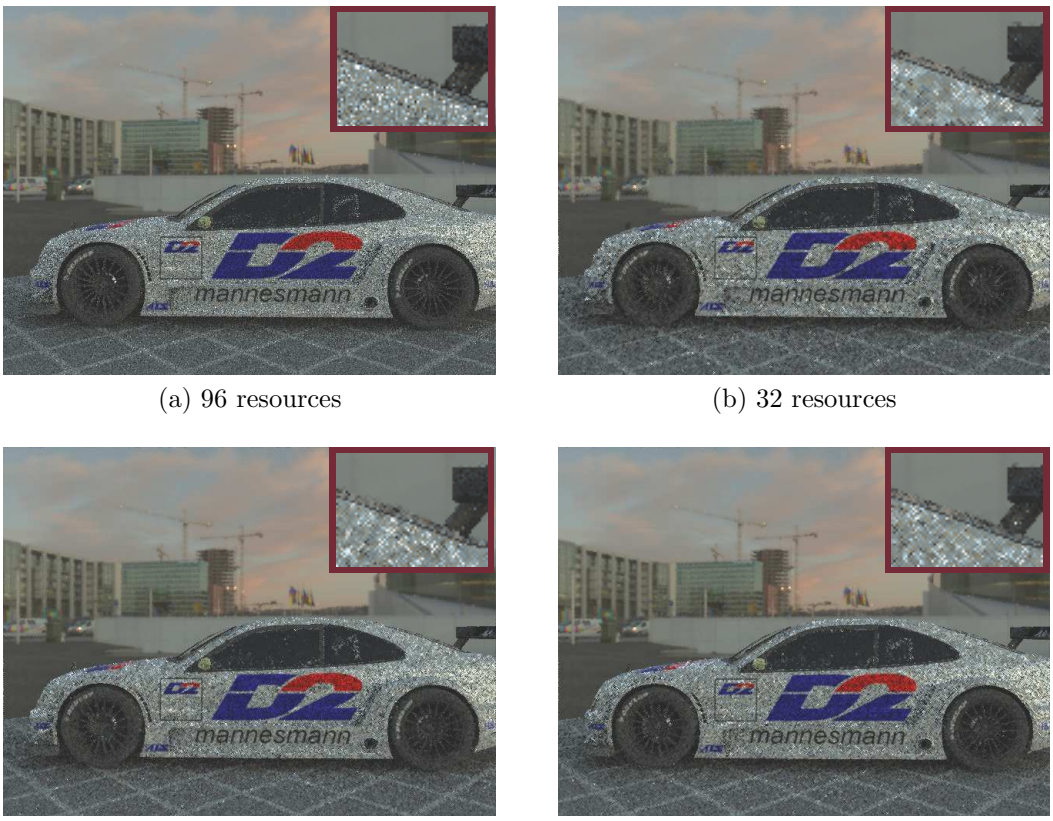


(c) Race Car



(d) Sponza

Figure 7.4: The comparison of visual quality of an interactive sequence with different resources for four scenes.



(a) 96 resources (b) 32 resources

(c) 64 resources (d) Variable resources

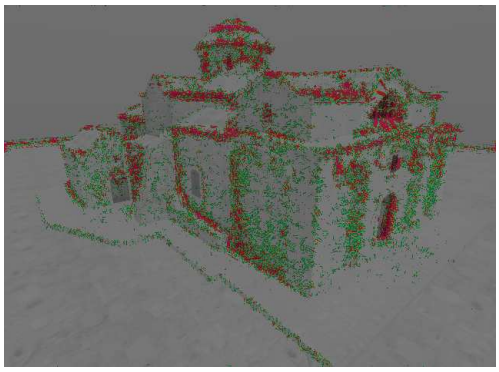
Figure 7.5: The frame 177 from the Race Car interactive sequence rendered on different number of resources. The inset is an enlarged portion of the image, showing the presence of more structured patterns due to reconstruction with lower number of resources.

The implementation was evaluated by comparing the visual quality of the scenes depicted in Figure 7.3, rendered on different number of resources. VDP was used for comparing the visual quality of different frames. Three kinds of visual comparisons are presented in the following subsections: one while interactively rendering the scene at a fixed frame rate, the second by rendering a static image with accumulation and the third comparing reconstruction quality for two hundred frames. For these comparisons, 96 identical processors were employed from the desktop grid to eliminate heterogeneity as a variable.

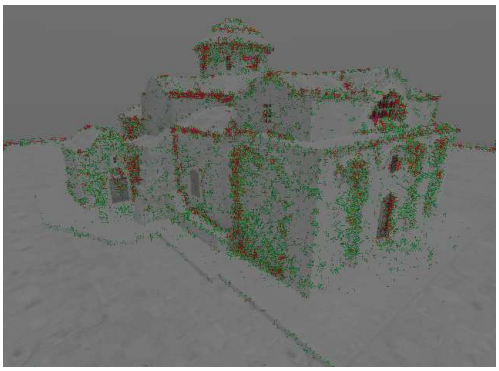
7.4.1 Interactive Sequence

In the first case, the same interactive sequence was rendered using different resources at a fixed frame rate (see Table 7.1). The graphs in Figure 7.4 compare

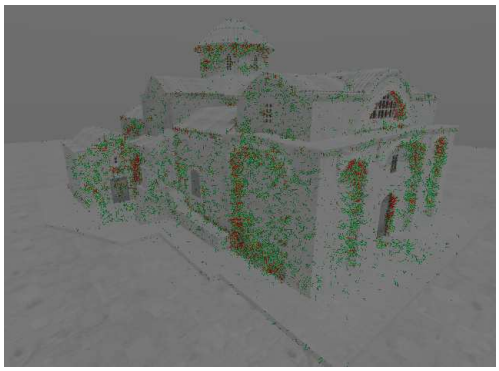
(a) Gold Standard rendered on 96 resources



(b) VDP output of frame rendered on 32 resources



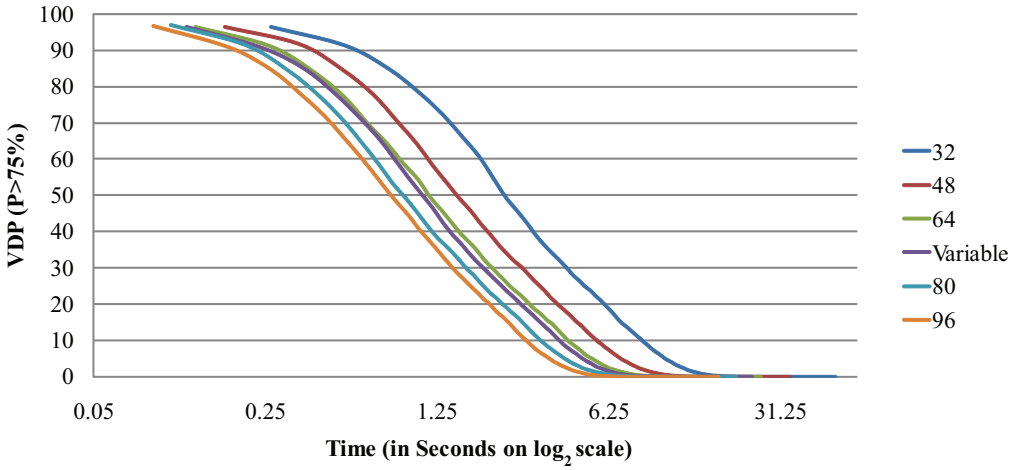
(c) VDP output of frame rendered on 64 resources



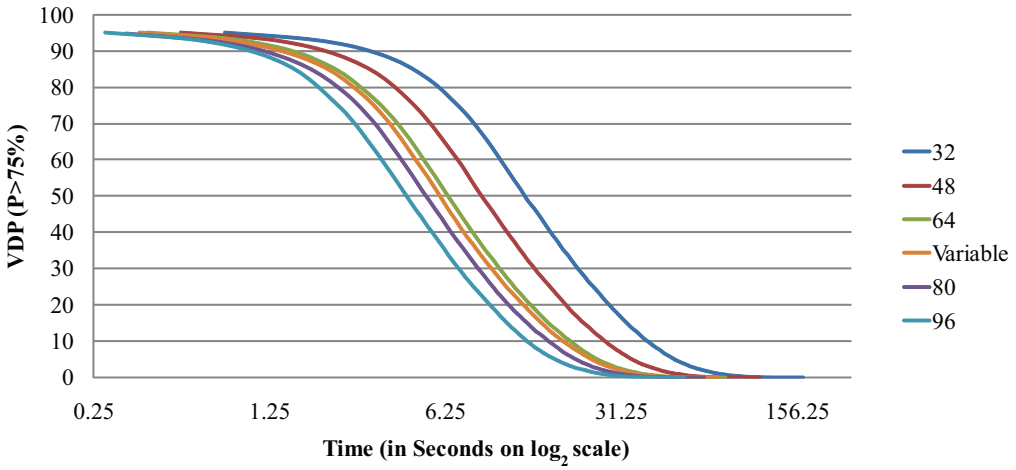
(d) VDP output of frame rendered on variable resources

Figure 7.6: The VDP comparison for frame 70 from the Kiti interactive sequence.

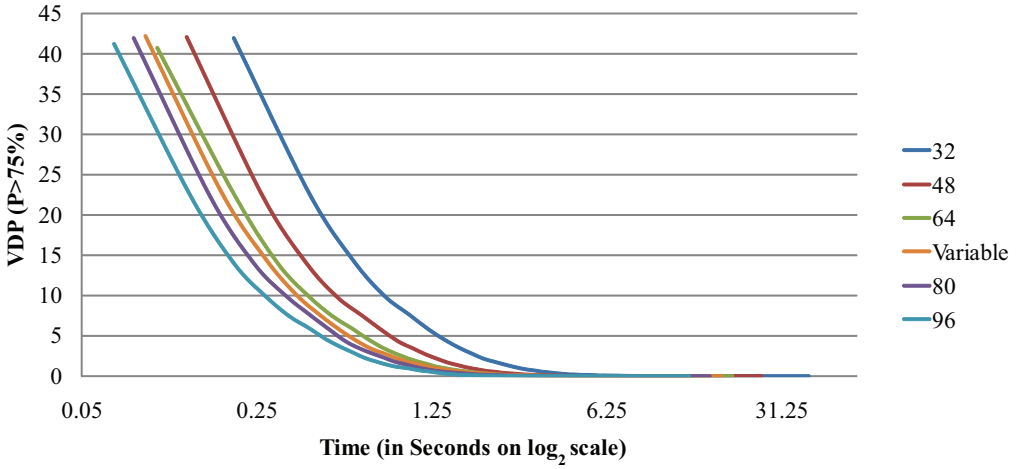
the VDP metric with a probability of detection greater than 75% for these sequences. The sequence rendered using 96 resources was used as the gold standard. These graphs show that, not surprisingly, with lower number of resources less result packets are received in the imposed time-constraint which means greater reliance on reconstruction to fill in the missing information as shown in the Figure 7.5. This deteriorates the image quality as expected and is confirmed by the VDP results (see Figure 7.6). For a predominantly indirectly lit scene such as Sponza, the VDP values are especially high. This is the result of the randomness introduced by path tracing only four samples per pixel for each update. As an indicator of the randomness, a data series is provided in Figure 7.4 comparing two sets of sequences both rendered with 96 resources, to serve as a reference. In addition to using fixed number of resources, a data series for variable resources is shown in Figure 7.4 where the number of resources varies per frame of the



(a) Cornell Box

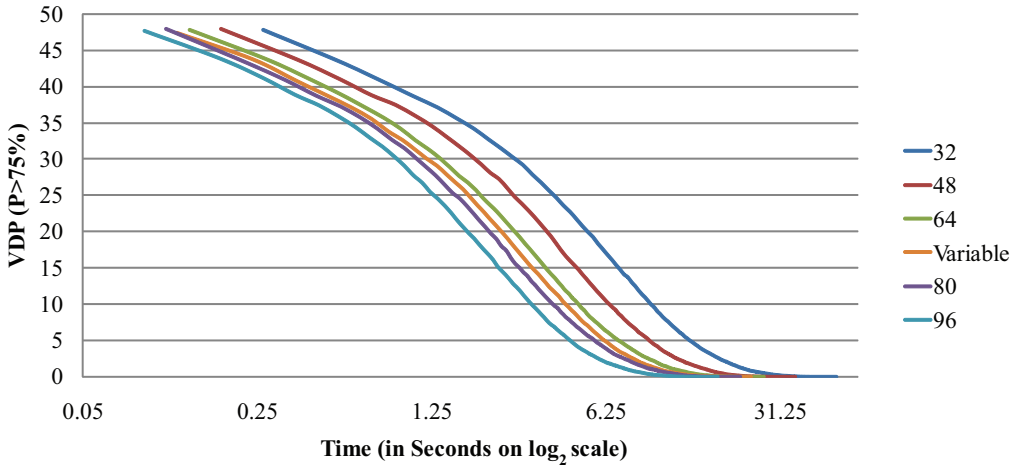


(b) Kalabsha

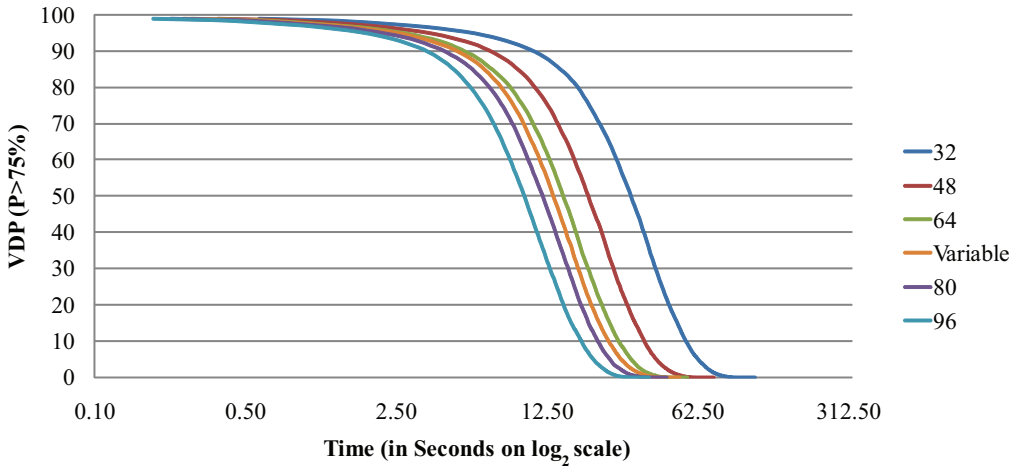


(c) Kiti

Figure 7.7: The convergence of visual quality of static images with different resources for five scenes.



(d) Race Car



(e) Sponza

Figure 7.7: The convergence of visual quality of static images with different resources for five scenes.

sequence. The resources, R at frame i are given by:

$$R_i = 96 - 24 \sin\left(\frac{2\pi i}{100}\right)$$

The variation of the resources was chosen to be sinusoidal to illustrate the algorithm's ability to compute at fixed frame rate with any number of resources. The system can handle sharp variations of resources as explained in Section 7.2.3, by skipping a frame in the worst case scenario.

Scene	Frames per Second
Cornell Box	6.66
Kiti	6.00
Race Car	5.00
Sponza	2.50

Table 7.1: Fixed frame rate used for various scenes.

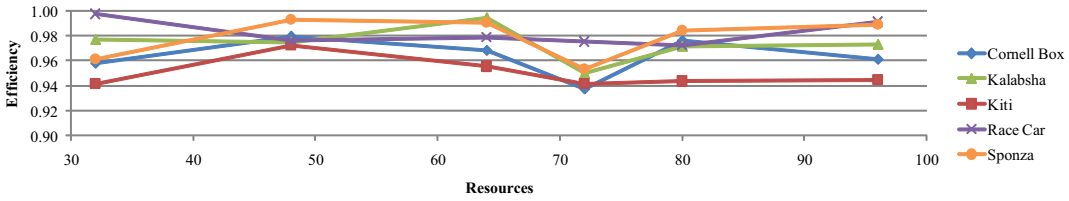
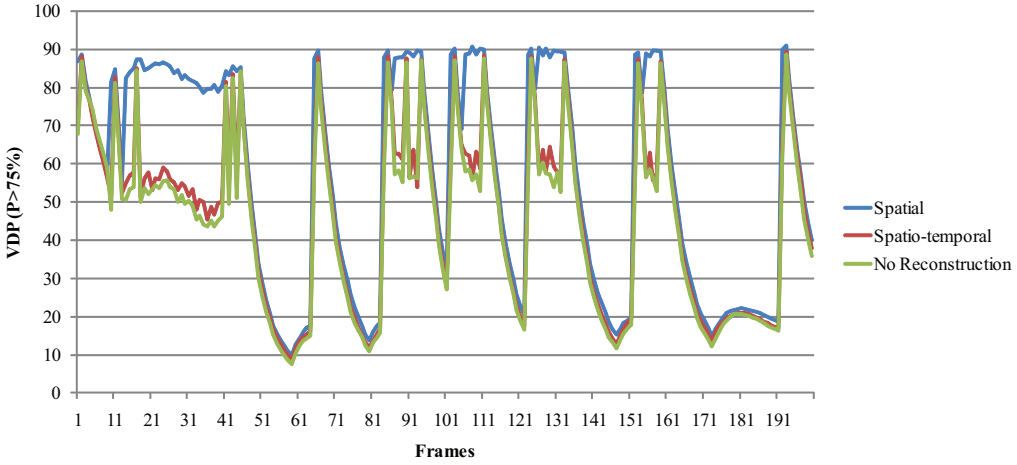


Figure 7.8: The efficiency comparisons for rendering on different number of resources.

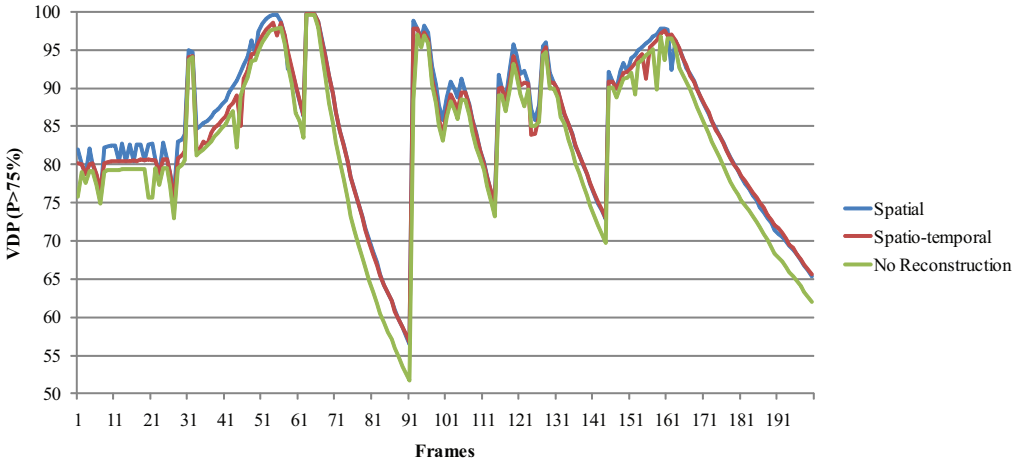
7.4.2 Static Image

In the second case, a single static image was rendered using different number of resources. The graphs in Figure 7.7 compare the VDP values showing the convergence of images with time. The converged image after 200 frames was used as the gold standard for evaluating the VDP metric. Even in the best case scenario (Figure 7.7c), it takes about a second for the VDP to fall below the 3% threshold where the two images are assumed to be the same. The variable resources data series uses the same function as before.

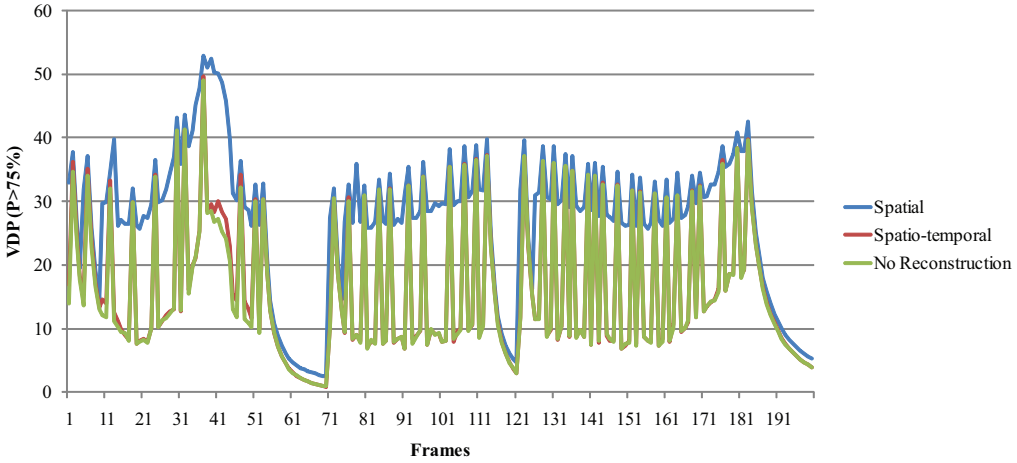
The efficiency of the interactive system was calculated by comparing the time it took for rendering these images with the time it took to render using a single processor, see Figure 7.8. It was found that the efficiency of the master did not decrease with an increase in number of workers on the test bed employed. The master running on a quad-core machine was not a bottleneck and with the use of non-blocking synchronisation, it scaled linearly on the test bed. The major bottleneck of the system was the rendering process. When the rendering cost was synthetically reduced to zero, frame rate of about 55-60 fps was obtained, which is an upper bound on the frame rate due to the available network bandwidth. When the master starts lagging due to too many workers, an approach similar to a hierarchy of masters can be used where the user would need exclusive access to more than one machine for running the masters. Also the load on the master can



(a) Cornell Box

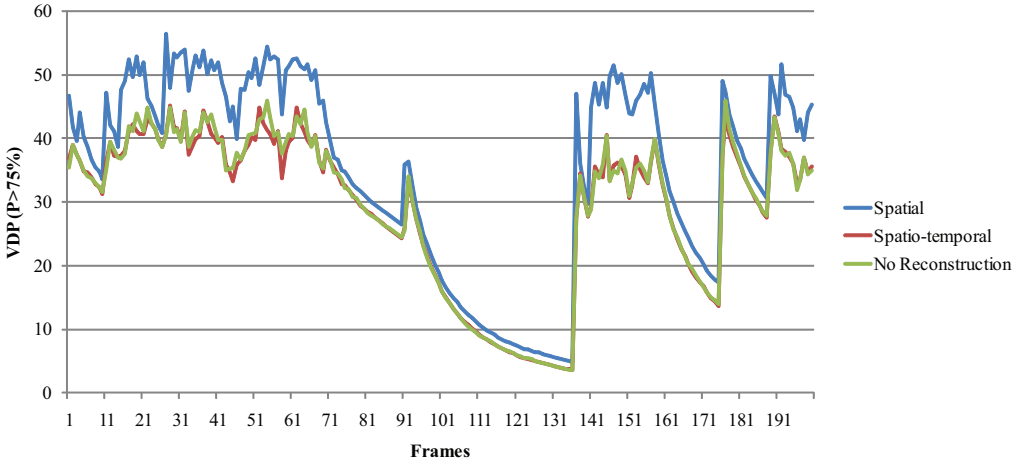


(b) Kalabsha

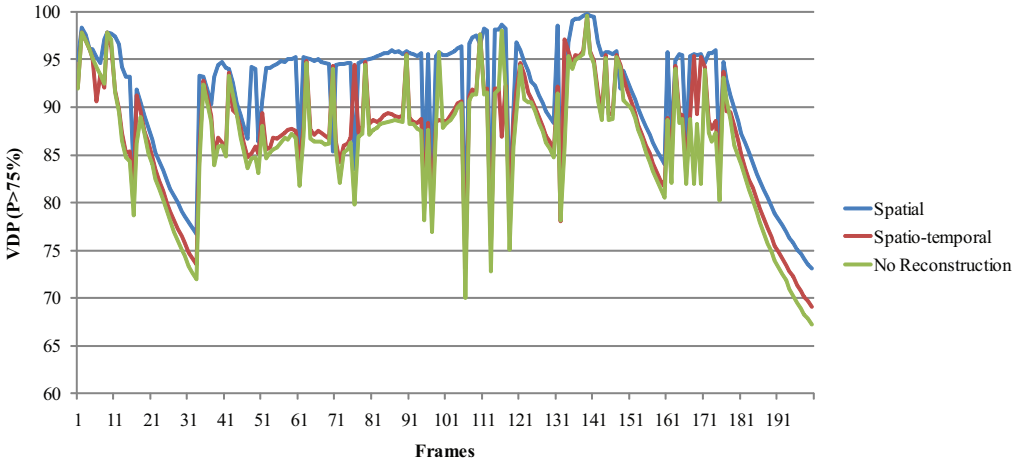


(c) Kiti

Figure 7.9: The visual quality comparison of reconstruction methods for interactive sequences.



(d) Race Car



(e) Sponza

Figure 7.9: The visual quality comparison of reconstruction methods for interactive sequences.

be reduced by decreasing the frequency with which the workers send requests to the master. This can be achieved by increasing the work load of the workers by calculating more samples per pixel in such a case.

7.4.3 Frame Reconstruction

Finally, the visual quality of the reconstruction was compared for interactive scenes. The graphs in Figure 7.9 show the advantage of using spatio-temporal reconstruction in contrast to spatial reconstruction only as used for static images in the previous chapter. The VDP plot for spatial reconstruction is similar to the

spatio-temporal reconstruction while the image accumulates. However, during the transition period while the user is interacting with the scene, the VDP for spatial reconstruction is considerably higher than the spatio-temporal reconstruction. During this period, the path tracing noise is especially high due to the low number of samples for each of the frames. Therefore to reduce this temporal noise, it is beneficial to use previous samples taking into account discontinuities for improving the visual quality of the frames. For these comparisons, frames with no missing pixels were used as the gold standard. Furthermore, a data series is plotted for each scene by comparing two sequences with no reconstruction to indicate the path tracing noise as before. The VDP plot of spatio-temporal reconstruction closely follows that of no reconstruction, indicating that the reconstruction method has an unnoticeable impact on the visual quality while employing the quality constraints specified in the Section 7.3. On the other hand, if a fixed frame rate is specified instead of using quality constraints, reconstruction artefacts are visible when lower number of resources are used, as shown in Figure 7.5. In such a case, to prevent deterioration of visual quality due to reconstruction artefacts beyond acceptable limits, either the fixed frame rate must be lowered or more resources must be employed.

7.5 Summary

This chapter described a novel fault-tolerant interactive rendering algorithm. This algorithm allows the users to achieve high-fidelity interactive rendering at a low cost using volatile computing resources. Desktop grids have been generally used for high-throughput computing, but this method demonstrates their potential for performing interactive visualisations. The heuristics employed by the method allowed successful monitoring of the variability in computational power to provide a smooth user interaction. Although the implementation used path tracing for calculating the image, other point-sampling based rendering algorithms can be similarly adapted for interactive visualisations.

The experiments conducted indicate that the rendering still remains the major bottleneck in the system and any advances which would make it faster would potentially improve the performance of the system. Recently GPU-based ray tracing engines [PBD*10] have emerged, which provide a significant speed-up over CPU-based ones. In the future, when the hardware for running such ray

tracing engines becomes more widespread, this method will naturally be able to exploit this hardware on idle machines to provide higher quality interactive rendering on desktop grids.

CHAPTER 8

Time-constrained Animation Rendering on Desktop Grids

This chapter extends the idea of combining sparse sampling and image reconstruction as a fault-tolerant mechanism for rendering animations in a time-constrained fashion on desktop grids. Two algorithms are presented and compared, which show that by employing multi-dimensional quasi-random sampling, the quality of the whole animation can be progressively enhanced. Also, this allows the system to become resistant to temporal variations in the computational power of the desktop grids and changes in the computational complexity across the frames of the animation.

This chapter is organised as follows: Section 8.1 introduces the chapter. Section 8.2 discusses the novel time-constraint fault-tolerant animation rendering algorithms. Their implementation is described in Section 8.3. A comparison between the two algorithms is presented in Section 8.4 and finally the chapter is summarised in Section 8.5.

8.1 Introduction

The variable nature of computing on desktop grids makes it difficult to estimate the time span of a computation. However, restricting the computation time makes the employment of desktop grids attractive in a production environment where deadlines have to be met. Furthermore, a fault-tolerant algorithm designed with the intent of creating the highest quality animation in a given time limit would be ideal for parallel animation rendering on variable resources. The traditional approach of scheduling frames independent of one another while rendering

animations in parallel, such as those presented in Chapters 5 and [CSL06], is not fault-tolerant. They would have to rely on conventional fault-tolerance strategies (see Section 3.4.1) which inhibit performance as described earlier in Section 6.1. In addition, animations are generally synthesised by either rendering a frame until it is finished or by spending a fixed amount of time on each frame, rather than imposing a time-constraint on the whole computation. Therefore, traditional approaches require modification for effectively employing desktop grids for time-constrained animation rendering. This is achieved by using the fault-tolerant mechanism devised for image/interactive rendering previously in Chapter 6 and 7 respectively. The imposed time-constraint for the presented approaches is less strict than that used for interactive rendering described in the previous chapter as the aim is to render at a higher quality. Hence, the reliance on reconstruction for algorithms presented in this chapter is minimal.

The rendering of animations on the desktop grids conforms to the conventional view of them as a high-throughput resource and therefore proper load balancing is important, as it has a significant effect for long running computations. Some frames of an animation may require more computational time than others due to the changes in scene complexity. Therefore, the computation needs to be load balanced across the frames in addition to load balancing on parallel resources when rendering towards a deadline. An unbalanced load would lead to undesirable differences in the visual quality of the different frames of the animation. This can be avoided by progressively updating the whole solution using multi-dimensional quasi-random sampling over the entire length of the animation.

8.2 Fault-tolerant time-constrained animation rendering

This section presents two fault-tolerant algorithms for rendering animations on variable resources in a user-specified time interval.

8.2.1 Straightforward Approach

A straightforward approach for rendering an animation under a time-constraint on a desktop grid would be to divide the time-constraint equally for each frame of the animation. The task then becomes to render all the frames with Equal Time-constraint Per Frame (ETPF) approach in a manner similar to the ones presented in Chapter 6. Each frame can be subdivided into sets of quasi-random pixels and

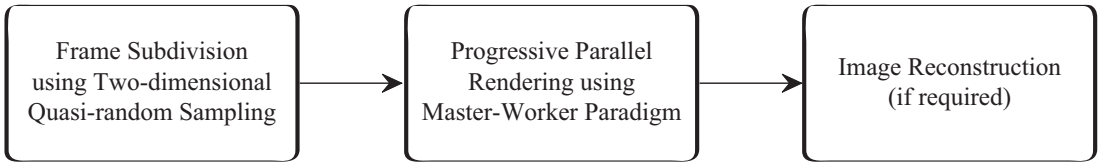


Figure 8.1: The pipeline for the ETPF approach. This pipeline is used for each frame of the animation which are time-constrained equally.

then rendered in parallel using the master-worker paradigm and reconstruction may be used in case of missing pixels. The pipeline for the ETPF approach is shown in Figure 8.1.

The ETPF approach has two major limitations. Firstly, it would be susceptible to temporal variations in computational power of the desktop grid. If the computational power of the desktop grid varies significantly for the duration of the total time-constraint imposed, then some frames would be rendered at a higher quality than the others resulting in uneven frame quality across the animation. Secondly, spending equal time for each frame would also result in non-uniform visual quality since the computational complexity can vary across different frames of an animation. Hence, to overcome these two limitations, a better load balancing strategy is required for rendering animations under time-constraints on variable resources.

8.2.2 Multi-dimensional Quasi-random Sampling Approach

An improved strategy for rendering animations under time-constraint on a desktop grid would be to use a Multi-dimensional Quasi-random Sampling (MQS) approach for subdividing the computations. This would entail that each job would consist of rendering pixels which would be spread not only on the image plane of a single frame, but they would be quasi-randomly selected across multiple frames as well. An animation can be considered as a volume of pixels which can be sampled using a three-dimensional quasi-random sequence as depicted in Figure 8.2. The modified pipeline for this approach is shown in Figure 8.3.

By quasi-randomly sampling in three dimensions, the two major limitations of the ETPF approach can be overcome. The MQS approach would be more robust to temporal fluctuations in computational power of a desktop grid in contrast to the ETPF approach, as it does not need to decompose the time-constraint for each frame and hence any pixel of the animation may be scheduled

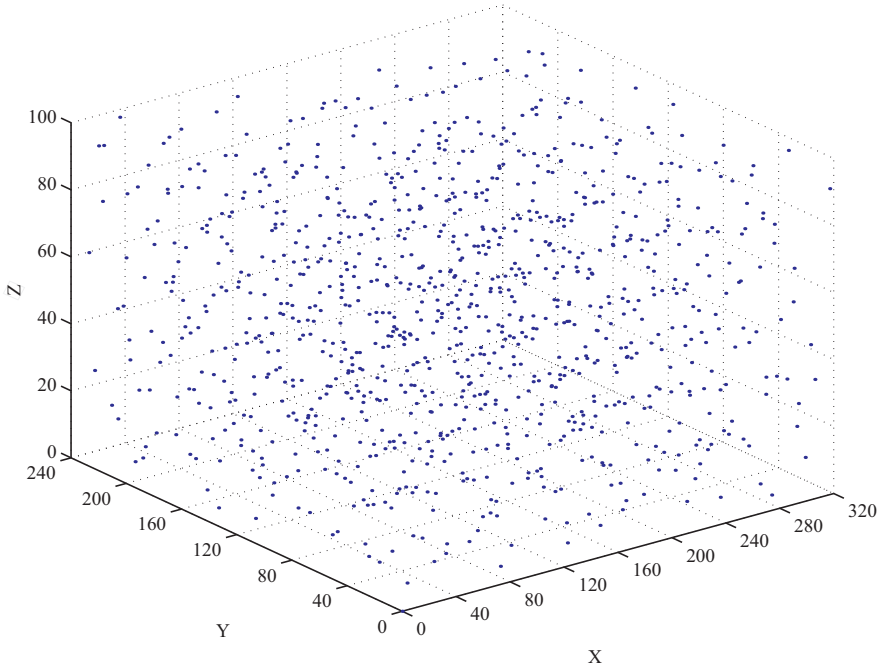


Figure 8.2: Three-dimensional Sobol Sampling

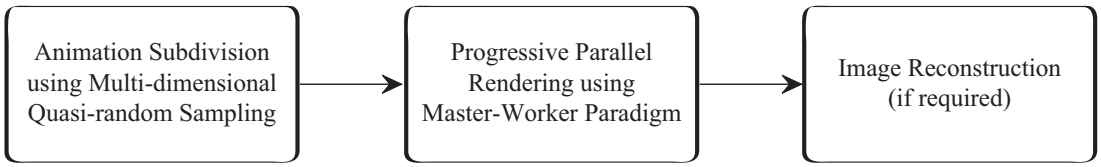


Figure 8.3: The time-constrained pipeline for the MQS approach.

at any given time. Also, it would achieve better load balancing by scheduling pixels from multiple frames simultaneously and therefore tackling the issue of variance in computational complexity across the animation. Furthermore, the MQS approach has another advantage over the ETPF. It progressively refines the whole animation and hence this gives the flexibility to stop and later continue the computation at any given time. In contrast, the ETPF approach employs a progressive rendering algorithm but the whole approach is not progressive as it tackles one frame at a time.

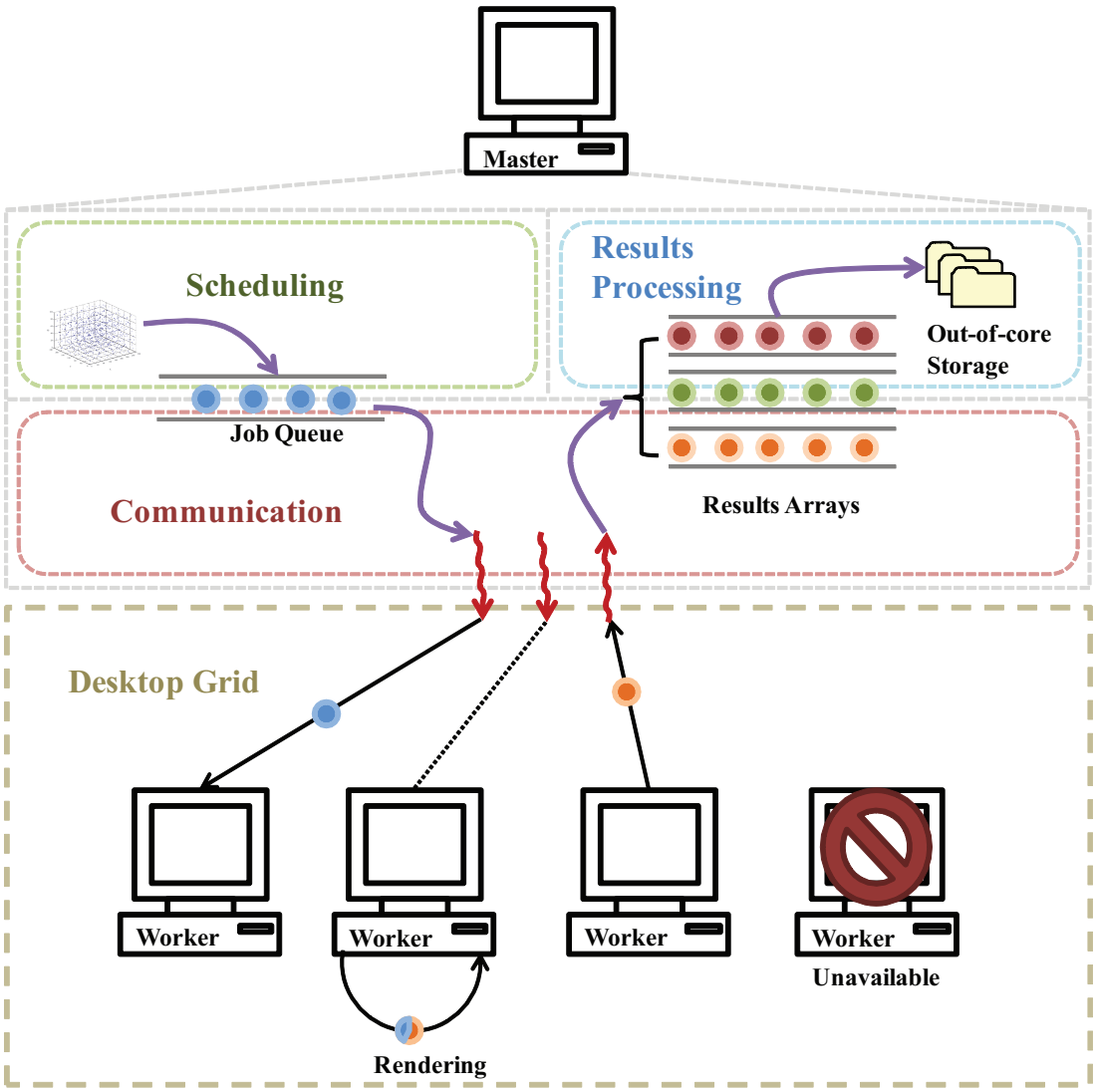


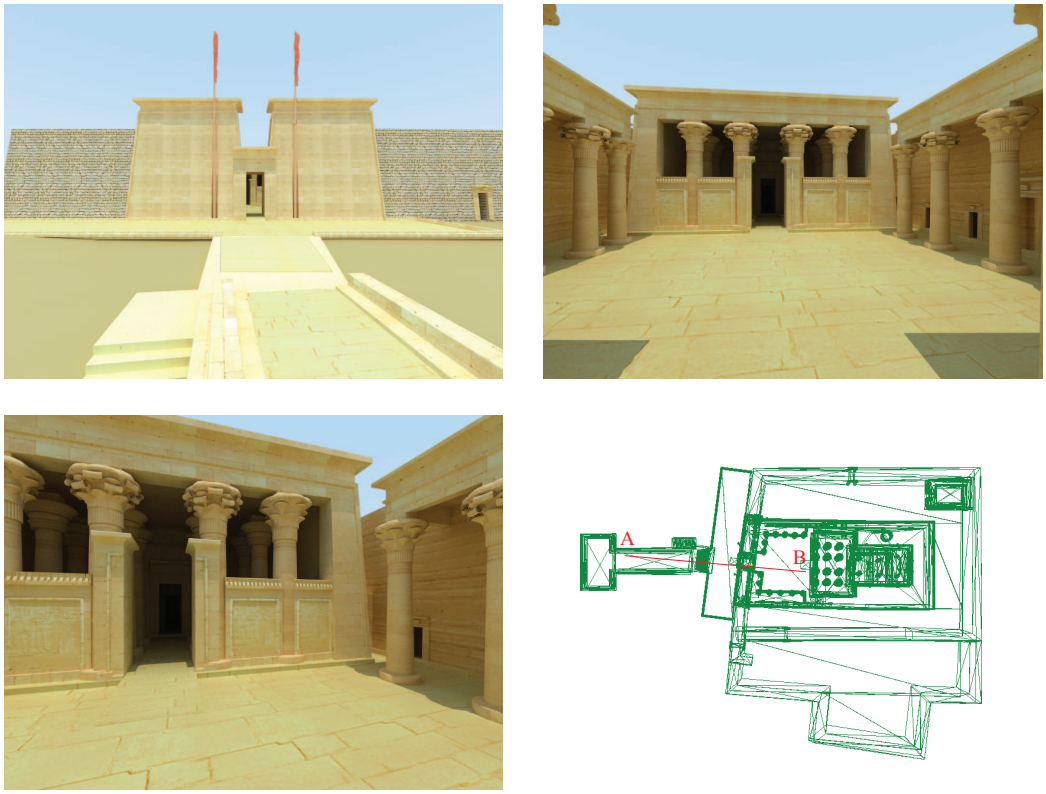
Figure 8.4: The overview of the time-constrained animation rendering system showing the interactions inside the master and between the master and the desktop grid.

8.3 Implementation Details

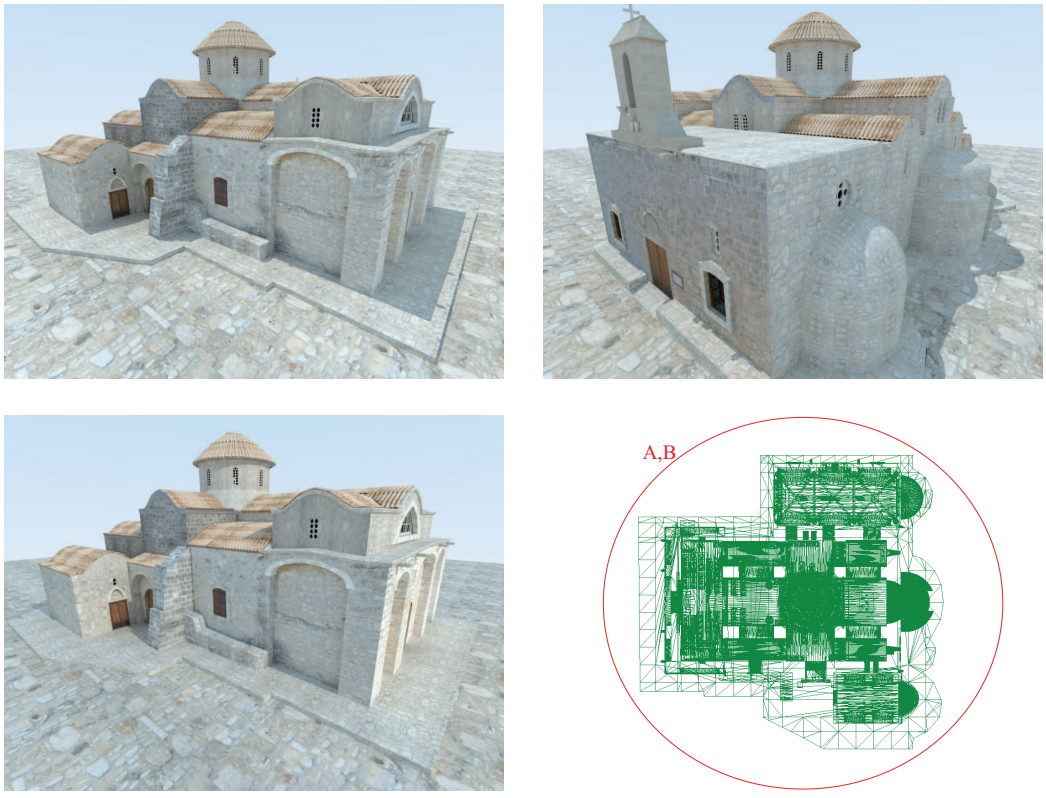
Both the algorithms mentioned in the previous section have been implemented and tested on the desktop grid described earlier in Section 7.3. As before, the results presented in the next section were obtained on 96 identical processors of the desktop grid to eliminate heterogeneity as a variable. The rendering was carried out by the workers using path tracing (see Section 2.6.2), however, other point sampling methods can also be employed. The animation frames were rendered at a resolution of 1024×768. The nearest neighbour algorithm (see Section 2.7.1) was used in the rare cases where reconstruction was required.

The implementation of the MQS approach needs to be carefully planned. The details for each pixel of the animation, such as pixel colour and number of samples computed, need to be stored in memory as any of the pixels of the animation may be processed at any given time. Even for a small animation of 720 frames with 1024×768 resolution, storing this data would require approximately 3.16 GB of memory. To overcome this potential problem, an out-of-core storage mechanism is necessary and memory mapped files were employed for this implementation. The component-based approach presented in Section 6.2.2 would further multiply the memory requirements, as data for each component would have to be stored separately. Hence, despite the fact that it may offer better visual quality for a single image, it is difficult to employ a component-based approach for rendering animations.

An overview of the system is depicted in Figure 8.4. First, the scheduler divides the computation into smaller jobs and places them on a job queue. Next, these are communicated to idle workers executing on the desktop grid when a request is received from them. They then process the job and send the results back. As pixels from multiple frames can be scheduled at the same time, multiple result arrays are used for storing the received data individually for each frame. A multi-threaded architecture is employed on the master to handle and prioritise the communication, such that the results are transferred to the out-of-core storage only when idle. This prevents the master from becoming a bottleneck in the whole process. A producer-consumer problem arises in the system as the workers produce the results while the master transfers (consumes) them to the out-of-core storage. Therefore, a balance between the rate of production and consumption is needed since the master has a limited memory space along with a managed synchronisation between the threads.

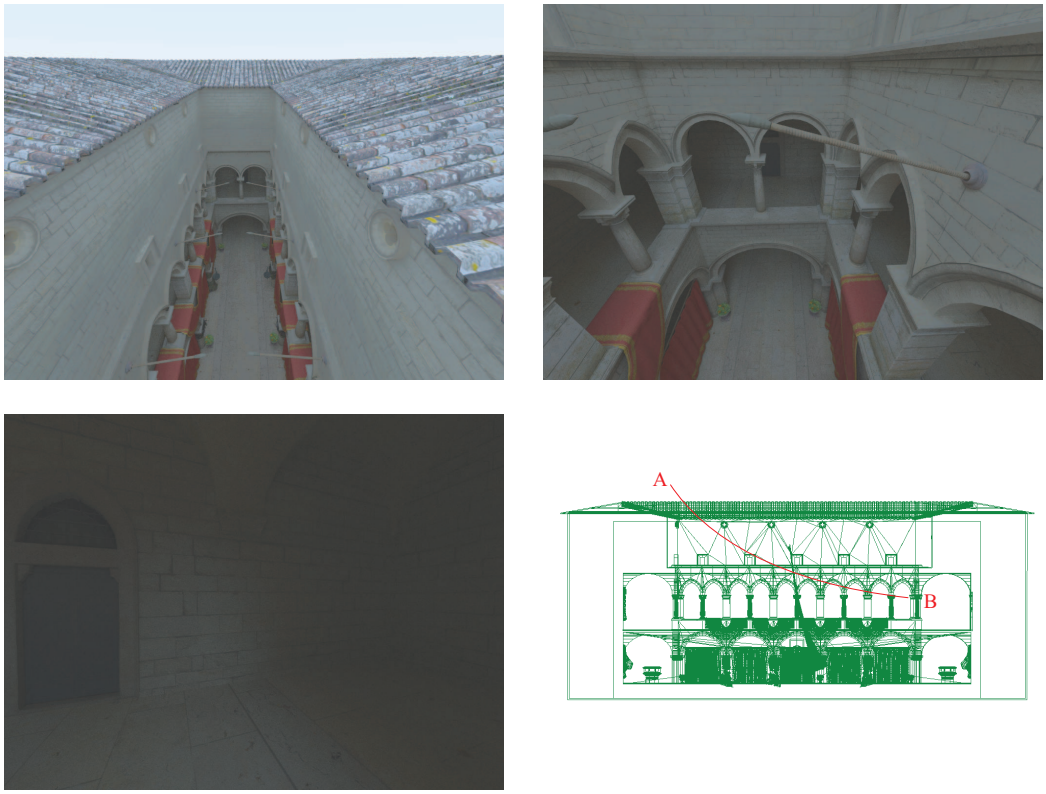


(a) Kalabsha



(b) Kiti

Figure 8.5: Start (top-left), middle (top-right), end (bottom-left) frames and the animation path (bottom-right) from A to B chosen for the three scenes.



(c) Sponza

Figure 8.5: Start (top-left), middle (top-right), end (bottom-left) frames and the animation path (bottom-right) from A to B chosen for the three scenes.

The scheduler needs to keep track of the time-constraint and monitor the job queue before adding more jobs based on one of the two approaches presented. For the ETPF approach, jobs were scheduled by splitting each frame quasi-randomly as earlier in Section 4.3, and rendering each frame separately for equal portions of the total time-constraint. The animation subdivision for the MQS approach was carried out using a three-dimensional quasi-random Sobol sequence. The concept for scaling and quantising a quasi-random sequence, as explained in Section 4.3, can be extended to three-dimensional sequence by using three scaling factors (one for each dimension) chosen in a similar manner. A three-dimensional base-2 Sobol sequence was used rather than using those presented in [KK02] as they are valid for a single-dimension only. However, a three dimensional base-2 sequence is fixed and cannot be changed as described in [KK02]. Hence, to increase the fault-tolerance, the sequence was shifted circularly to obtain a different grouping of set of pixels between iterations.

Scene	Time-constraint (minutes)	Number of Frames
Kalabsha	360	720
Kiti	120	720
Sponza	150	240

Table 8.1: Time-constraints used for various animations

8.4 Results

The two time-constrained rendering approaches were compared for the three animation sequences depicted in Figure 8.5. The computational complexity of the Kiti animation is fairly constant across the animation while it varies substantially for both the Kalabsha and the Sponza animations. The time-constraints used for rendering and the number of frames for the animations are listed in Table 8.1. The time-constraints were chosen to be approximately 10% of the time it took to render the reference animation on the desktop grid.

Three types of fault variations were used for comparisons: no faults (NF), random faults (RF) and temporal faults (TF). The whole desktop grid was dedicated to render the animations for the NF condition. The unpredictable nature of shared resources at run-time on a desktop grid was modelled by using RF. In order to simulate RF condition, a result sent to the master was rejected with a probability of either 25% (RF25) or 50% (RF50). This was achieved by employing a Mersenne Twister pseudo-random number generator [MN98]. Finally, to mimic the time-variant nature of desktop grids TF was used. For the first half of the time-constraint, 25% random faults were generated while for the second half 75% random faults were generated. This is depicted in the graph shown in Figure 8.6. On an average, this is similar to RF50 and hence the notation TF50 is used.

8.4.1 Visual Quality

The visual quality of animations computed with the two approaches under different fault variations was measured using the VDP metric. A high-quality reference animation was calculated with 1000 SPP and no time-constraints or faults and it was used as the gold standard for computing the VDP results for each frame. The results have been presented in Figure 8.7. It can be seen from these results that MQS-NF has the best visual quality as expected. The VDP values increase

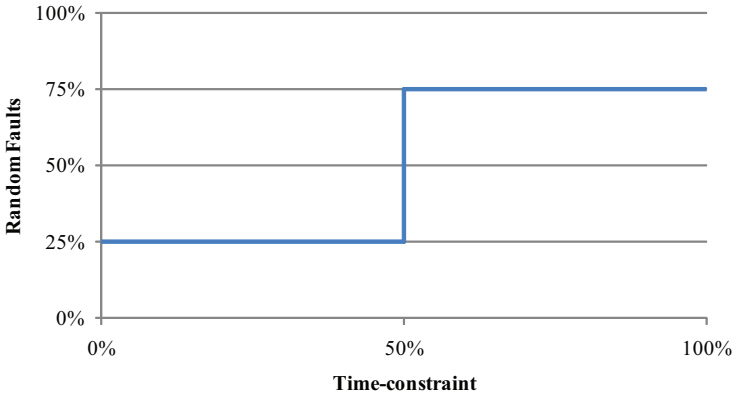
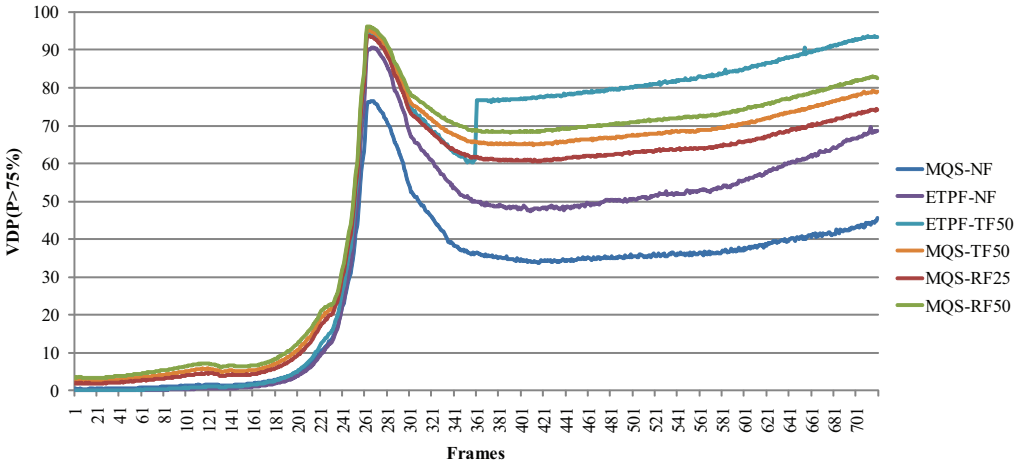


Figure 8.6: The Temporal Fault variation (TF50)

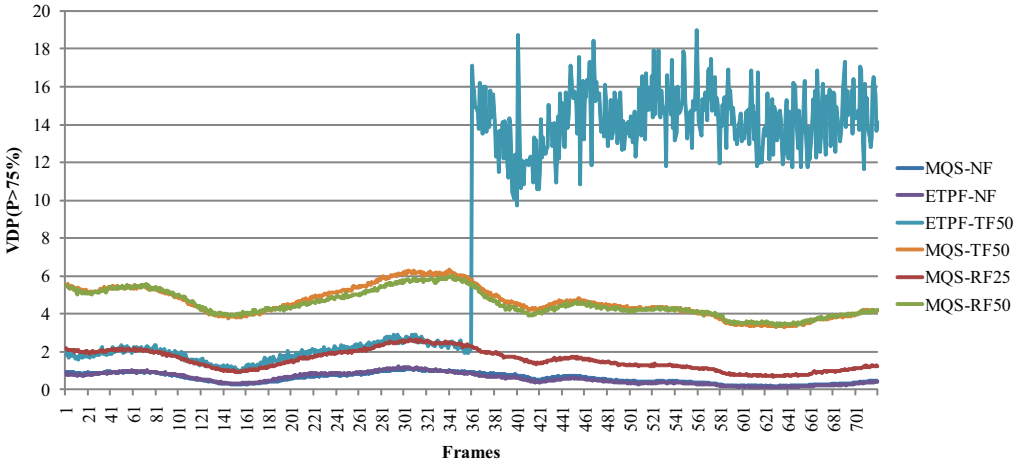
for the three cases of MQS: MQS-NF, MQS-RF25 and MQS-RF50, as the number of faults generated increase. An illustration comparing the algorithms under different fault models is presented in Figure 8.8.

The VDP curve for ETPF-NF is very similar to MQS-NF for the Kiti animation since its computational complexity does not vary significantly across the frames. However, for the Kalabsha and the Sponza scenes, ETPF-NF performs much worse than MQS-NF for the computationally difficult frames. The difference between the two algorithms is much higher for the TF50 condition. MQS-TF50 and MQS-RF50 have very similar VDP plots showing that the MQS approach can resist temporal fluctuations of computational power. On the other hand, ETPF-TF50 is similar to MQS-RF25 for the first half of the time-constraint and hence better than MQS-TF50 for those frames. However, for the second half of the time-constraint, the VDP curve for ETPF-TF50 shows a drastic increase, while MQS-TF50 doesn't depict any such phenomenon. Hence, the MQS approach aims at obtaining an even visual quality across the animation while the ETPF approach is susceptible to uneven visual quality especially in presence of temporal variations.

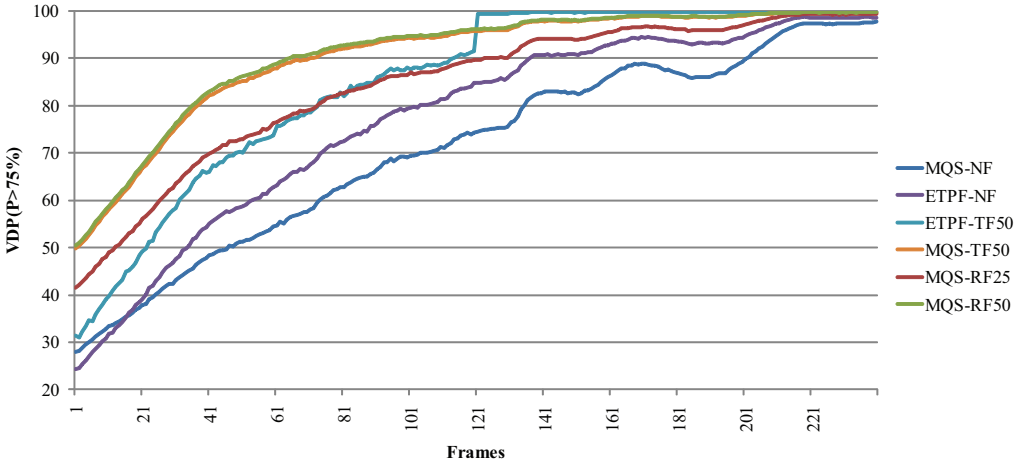
The VDP values for the difficult frames of the animations are considerably high. This is due to the fact that the time-constraint used for the calculation of these values was only 10% of the time it took to render the reference animations and some computations were discarded due to simulated faults. The latter half of the Sponza animation has especially high VDP values making it difficult to compare the various conditions described above for these frames. To overcome this limitation in the VDP graphs, the Root Mean Square Error (RMSE) quality



(a) Kalabsha

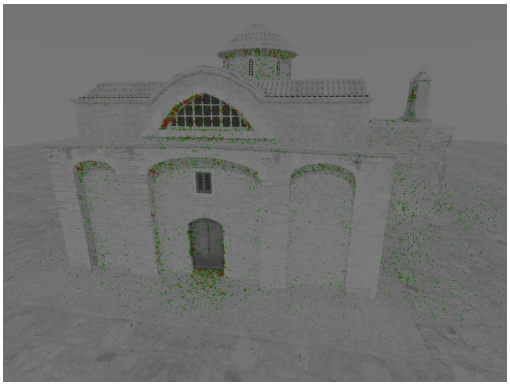


(b) Kiti

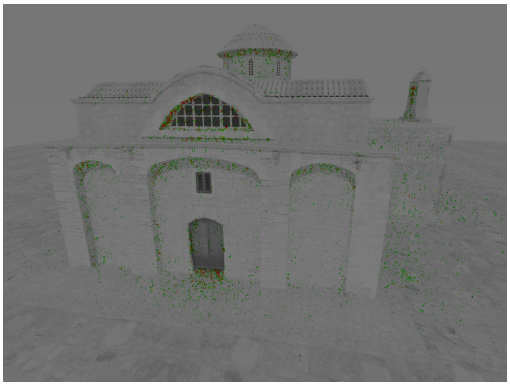


(c) Sponza

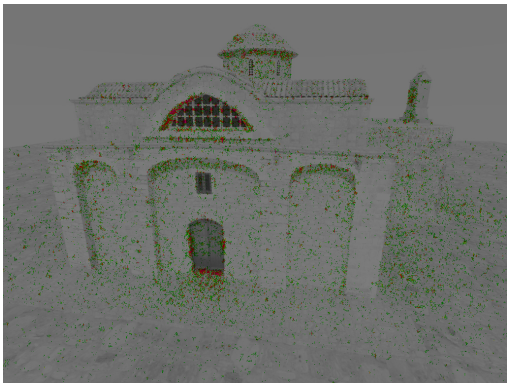
Figure 8.7: The VDP comparisons for the animations rendered with two algorithms under different fault models



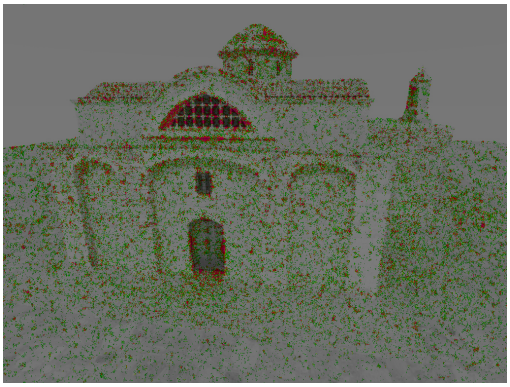
(a) MQS-NF



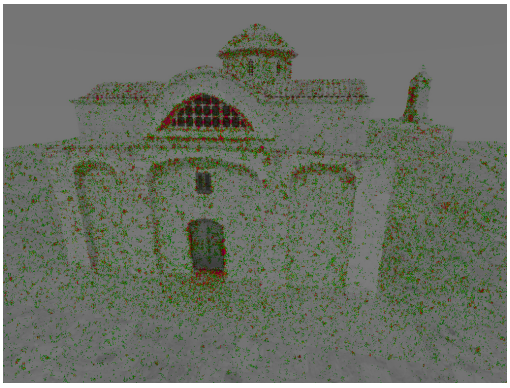
(b) ETPF-NF



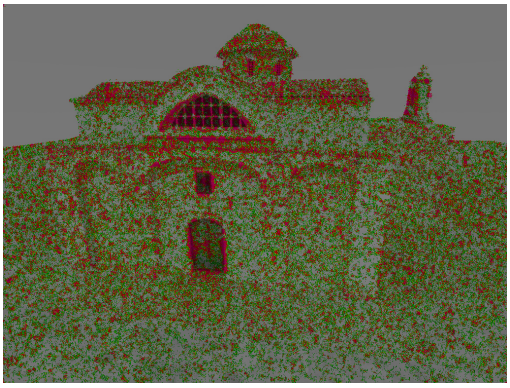
(c) MQS-RF25



(d) MQS-RF50



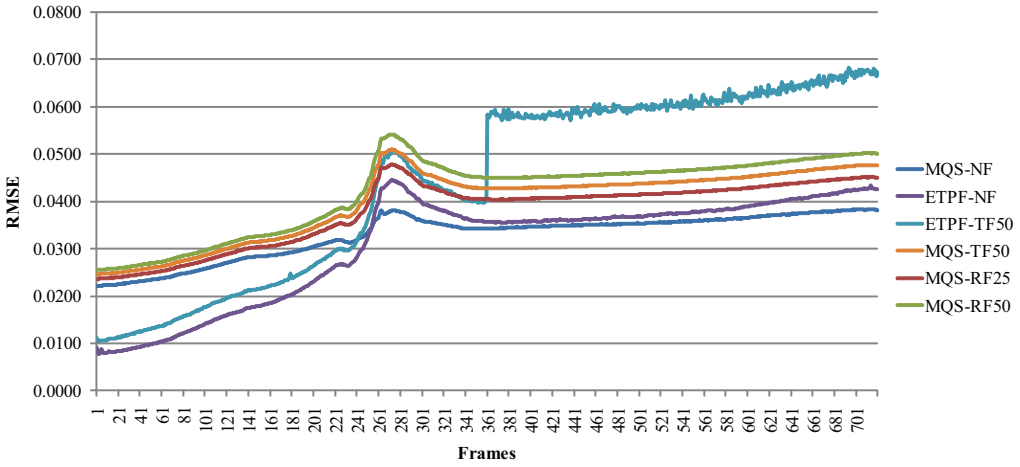
(e) MQS-TF50



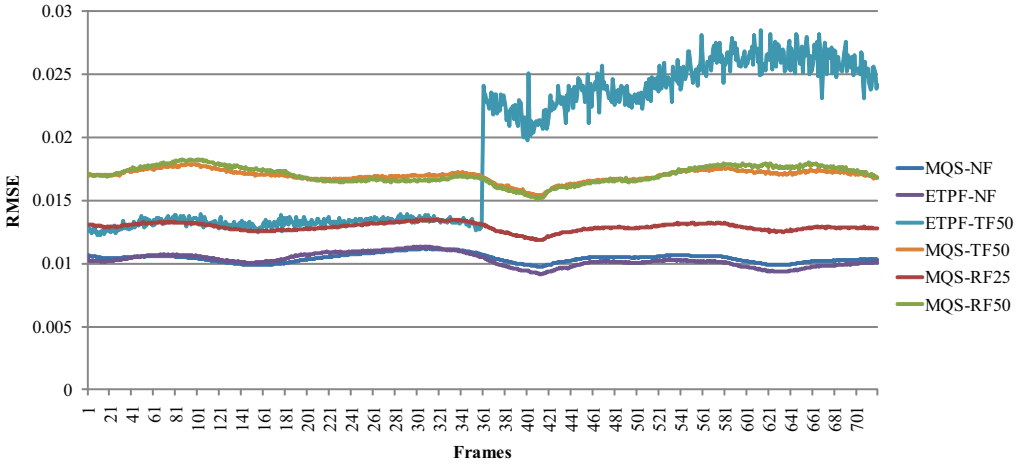
(f) ETPF-TF50

(g) Reference frame

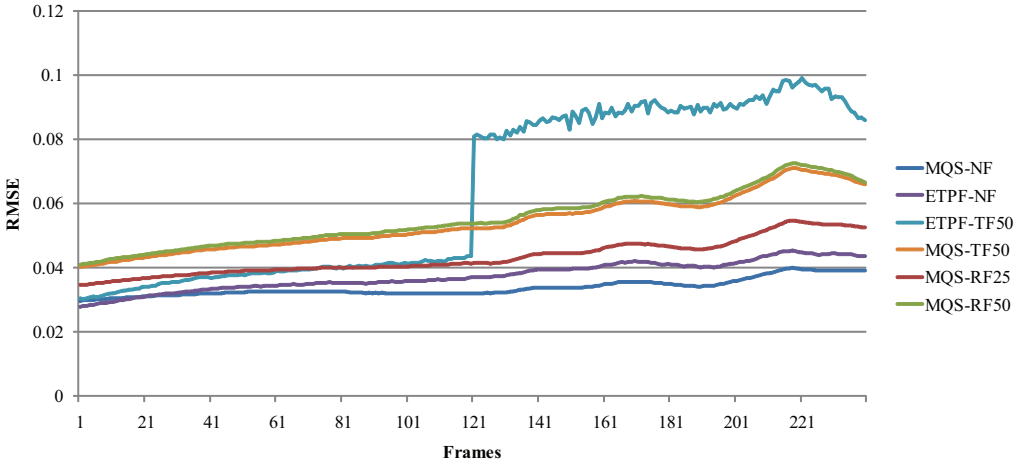
Figure 8.8: The VDP comparison for Frame 600 of the Kiti animation rendered with two algorithms under different fault models



(a) Kalabsha



(b) Kiti



(c) Sponza

Figure 8.9: The RMSE comparisons for the animations rendered with two algorithms under different fault models

Scene	Kalabsha	Kiti	Sponza
MQS-NF	0.0309	0.0102	0.0337
ETPF-NF	0.0309	0.0103	0.0374
MQS-TF50	0.0399	0.0170	0.0540
ETPF-TF50	0.0444	0.0189	0.0636
MQS-RF25	0.0379	0.0129	0.0432
MQS-RF50	0.0419	0.0170	0.0553

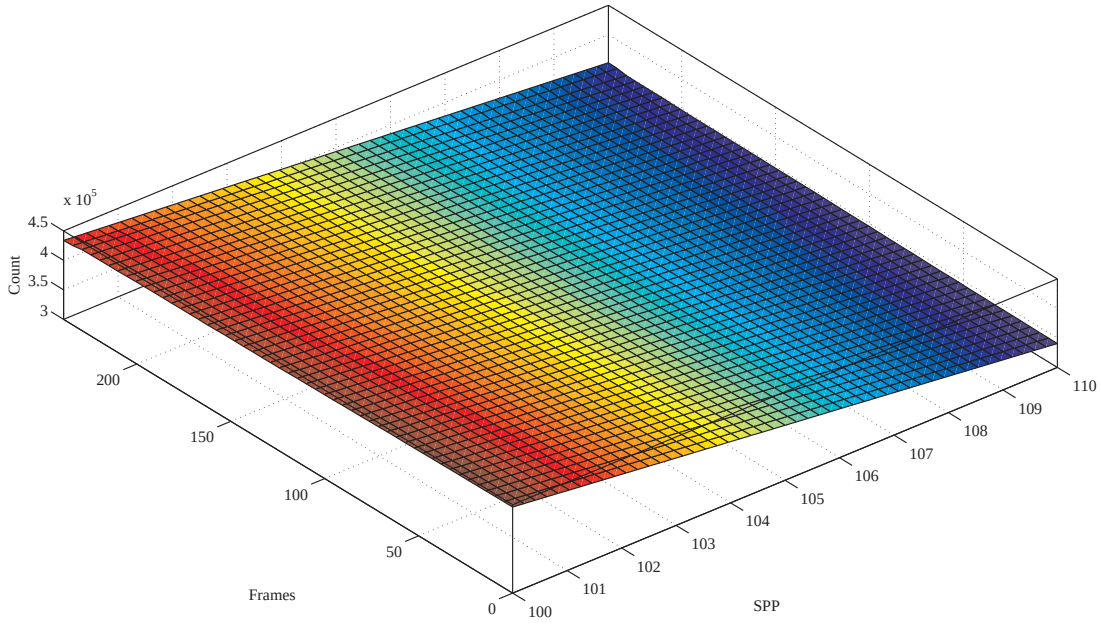
Table 8.2: Average RMSE Values

metric was also plotted and is shown in Figure 8.9. The RMSE graphs also support the inferences presented above based on the VDP graphs. The average RMSE values for all the animations are presented in Table 8.2. The MQS and the ETPF approach have similar average RMSE for the NF condition while the MQS is at least 10% better than the ETPF for the TF50 condition. The average RMSE increases for the MQS approach as expected, while the faults increase from NF to RF25 and RF50.

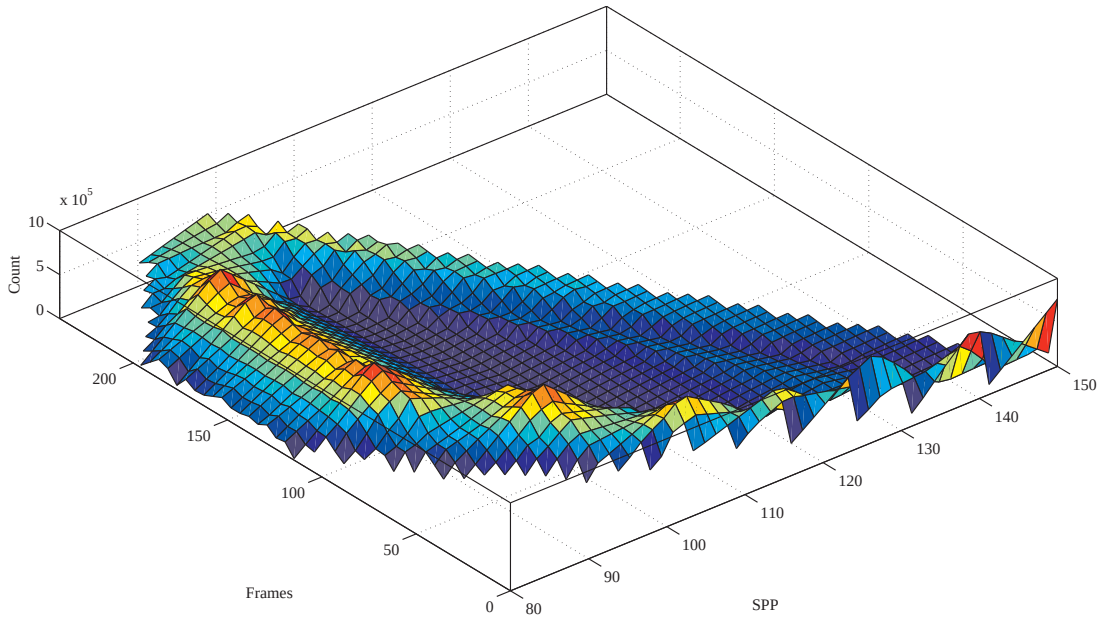
8.4.2 Fault-tolerance

The amount of work completed within a given time-constraint for a path tracing based renderer can be measured by the number of samples computed per pixel (SPP). Due to the quasi-random sampling employed in the two algorithms described in this chapter, SPP can be different for each pixel of the animation. The number of pixels (count) in a frame for which a given SPP have been computed, serves as an indicator of the load balancing of an algorithm in the presence of faults. The graphs between count and SPP for each frame of the Sponza animation have been shown in Figure 8.10 and indicate the fault-tolerant properties of the presented algorithms.

The graph in Figure 8.10a shows the progressive nature of the MQS approach. The MQS-NF plot shows that all the pixels in the animation have been calculated at a minimum of 100 SPP, and some of them have been further refined to 110 SPP, resulting in a smooth planar plot across the frames. However, the variation of the plot across the animation frames for ETPF-NF (see Figure 8.10b) depicts that the level of refinement for the ETPF approach is affected by the computational complexity of the frames and hence it calculates unequal SPP for different frames. The variation of SPP versus count for a frame for the MQS

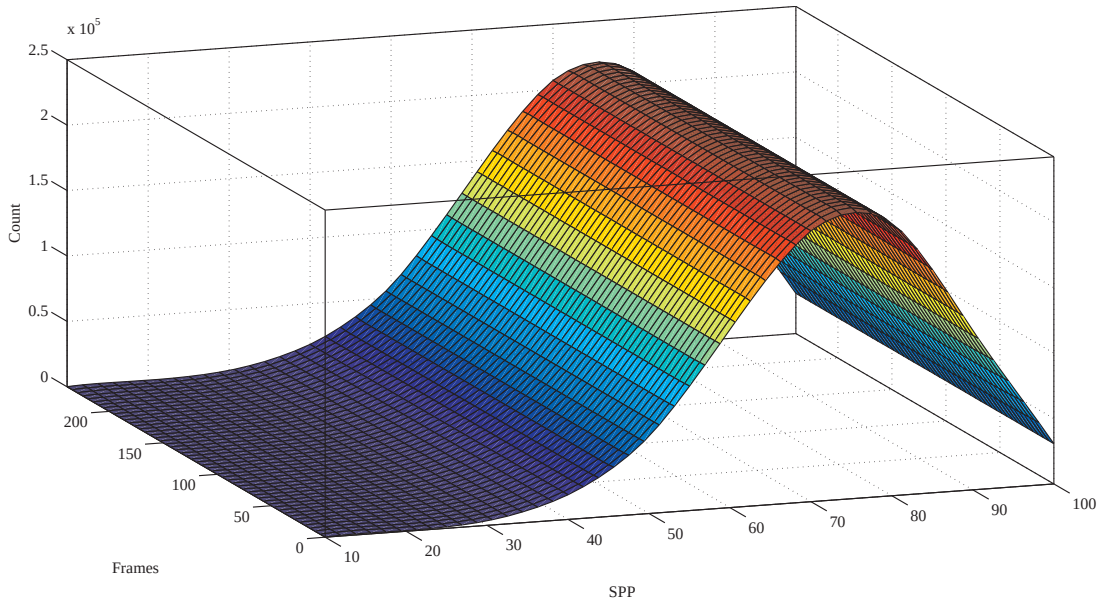


(a) MQS-NF

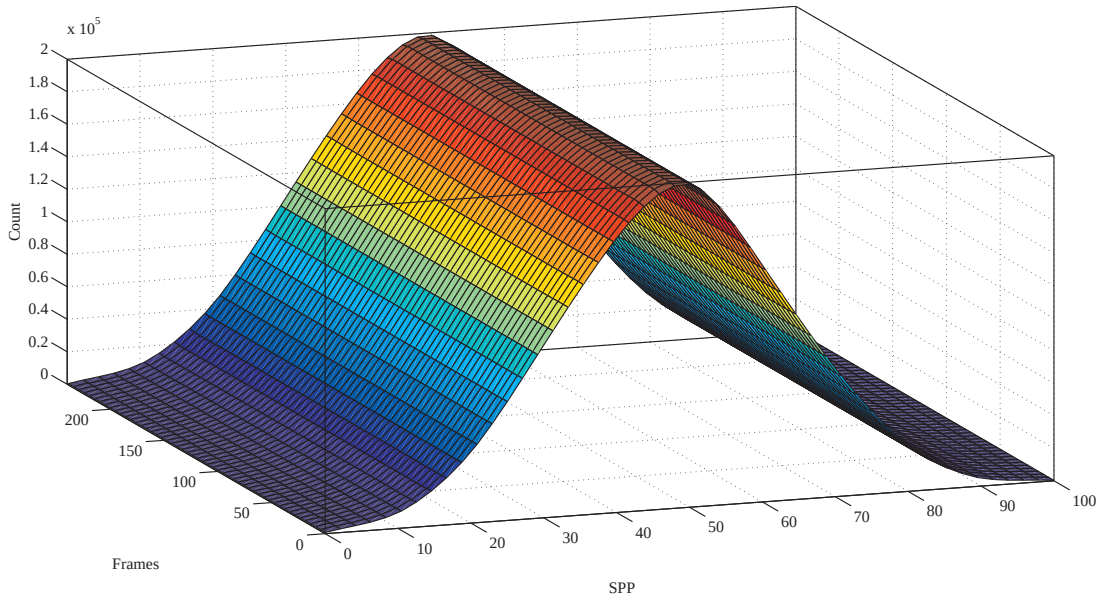


(b) ETPF-NF

Figure 8.10: Count versus SPP for different frames of the Sponza animation rendered with the two algorithms under various fault models

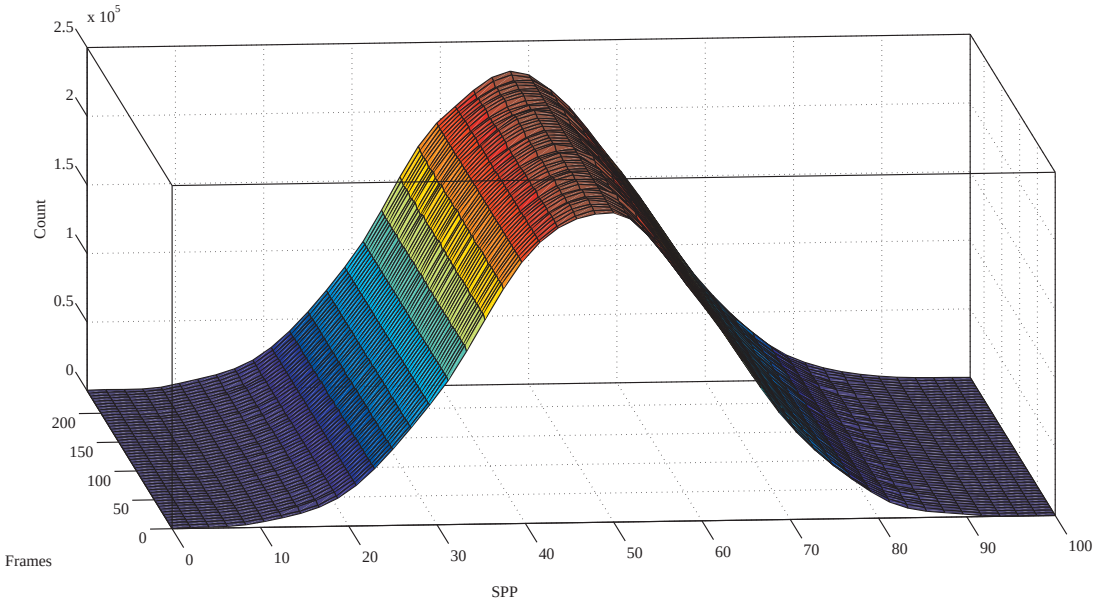


(c) MQS-RF25

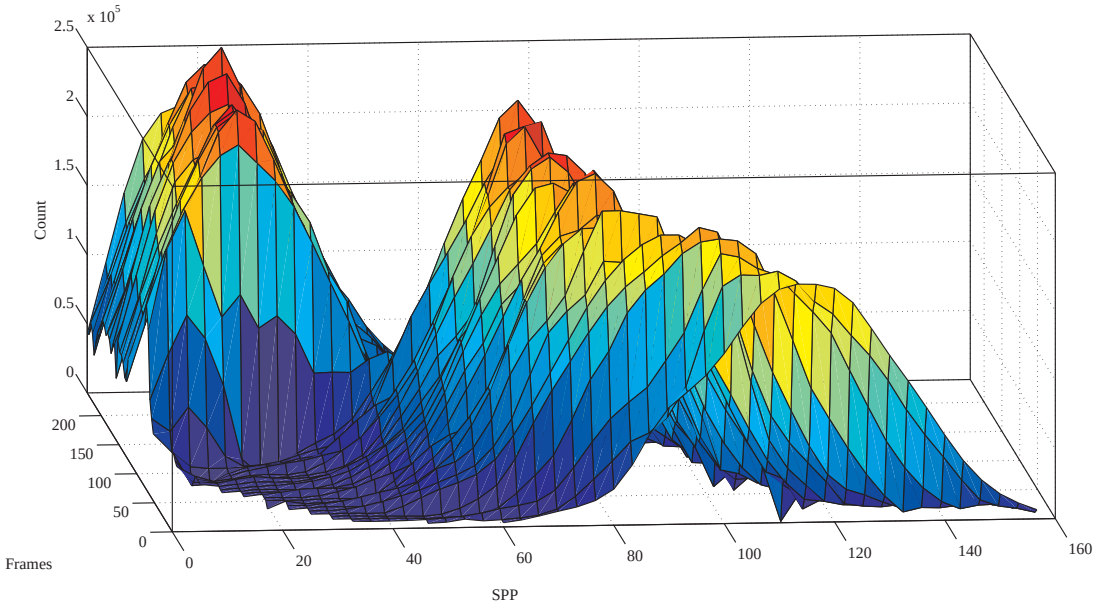


(d) MQS-RF50

Figure 8.10: Count versus SPP for different frames of the Sponza animation rendered with the two algorithms under various fault models



(e) MQS-TF50



(f) ETPF-TF50

Figure 8.10: Count versus SPP for different frames of the Sponza animation rendered with the two algorithms under various fault models

approach, in the presence of faults, follows a Gaussian distribution (see Figures 8.10c, 8.10d, 8.10e). Also, this variation is constant across the frames of the animation. As the number of faults increases, the peak of the Gaussian curve shifts leftwards illustrating the fact that less work is completed. Once again the graphs (Figures 8.10e and 8.10f) for MQS-TF50 and ETPF-TF50 show the effect of temporal variation. ETPF-TF50 shows two distinct lobes for the two halves of the time-constraint with different number of faults while MQS-TF50 remains unaffected by temporal fault variations and is similar to MQS-RF50.

8.5 Summary

This chapter presented two novel approaches for time-constrained rendering of animations on desktop grids. The results obtained showed that the MQS approach achieved better load balancing than the ETPF approach, while progressively refining the animation in case of faults. The MQS approach tackled the two flaws of the ETPF approach effectively, that is it was insensitive to both temporal variations of computational power of a desktop grid and difference in computational complexity across the animation frames.

The MQS approach presented in this chapter used three-dimensional quasi-random sampling for job subdivision. For simplicity, this approach assumes that each pixel of the animation contributes equally to the visual quality of the animation. This assumption can be removed by sampling a function which maps the pixels to their contribution to the visual quality as a fourth dimension, while quasi-randomly selecting the pixels. However, as this function would change while the rendering progresses it would have to be evaluated on the fly. This would both raise the memory requirements of the MQS approach as well as increase the complexity of the sampling process.

CHAPTER 9

Conclusions and Future Work

This chapter concludes the work presented in this thesis and presents the novel contributions and their possible impact. It also describes the limitations and potential directions for the future work to further explore the conjunction of high-fidelity rendering and shared computational resources.

9.1 Conclusions

The aim of this thesis was to accelerate high-fidelity rendering using inexpensive shared parallel resources as opposed to traditional expensive dedicated parallel systems. This was achieved by developing and testing novel fault-tolerant rendering algorithms on shared resources.

A two-pass algorithm for rendering animation sequences, using irradiance cache on computational grids, gained a speed-up over the traditional single-pass approach, due to the savings made in the computation of irradiance values. For example, the two-pass approach took less than half the time taken by the single-pass approach to render the Corridor animation, as shown in Table 5.1. Also, the visual quality of the animations was enhanced while using the two-pass approach (see Figure 5.3), since a temporally coherent irradiance cache was used for rendering successive frames. This study provided an insight into the current state of grid computing and the challenges faced while porting the irradiance cache algorithm to it. The current computational grids provide restricted access and are suitable for long running jobs with limited communication overheads.

The challenge of adapting rendering computations at run-time for exploiting variable resources of easily accessible desktop grids was overcome using a new fault-tolerant mechanism based on quasi-random sequences and image recon-

struction techniques. This fault-tolerant strategy did not curb the performance, in contrast to the conventional fault-tolerant strategies which are used for parallel computing on desktop grids. It was shown that, of the two time-constrained offline rendering algorithms which were developed using the novel fault-tolerant strategy, the component-based approach achieved better visual quality than a straightforward approach (see Figure 6.9). This was a result of subdividing the computations with finer granularity. However, these techniques took a few seconds to render the images and it was not possible to achieve interactive rates since these algorithms employed a traditional grid middleware for job scheduling which is not suited for this purpose.

Interactive rendering on desktop grids was made possible by using the novel fault-tolerant strategy and developing a job management and scheduling system. This fully dynamic system allowed the users to interactively change the virtual scene by modifying object positions, material properties, lighting conditions and camera attributes. Such a system had previously been restricted to only dedicated clusters or supercomputers. The heuristics which were developed, enabled the system to monitor the variable computational power of the desktop grid. This allowed adaption of the computations at run-time to provide the user with a fixed frame rate or visual quality. The system was able to achieve a fixed frame rate of up to 3 frames per second for the highly complex Kalabsha scene (861k polygons), while it provided 10 frames per second for a simpler Kiti scene (243k polygons). A spatio-temporal image reconstruction mechanism was also developed which helped in generation of temporally coherent images from sparse samples in real-time. This system demonstrated the use of desktop grids as a high-performance computing resource.

Finally, the fault-tolerance mechanism was enhanced for rendering animation sequences within a user-defined time-constraint by using multi-dimensional quasi-random sampling. The MQS approach performed better than the ETPF approach due to the fact that the MQS approach was unaffected by temporal variations in computational power and the variability in computational complexity of the frames of the animations, as shown in Figure 8.9. The average RMSE values for the MQS approach were at least 10% better than the ETPF approach in the presence of temporal faults (see Table 8.2). This was possible as multi-dimensional sampling allowed the quality of the whole animation to be progressively refined even in the presence of faults. These algorithms enabled the employment of shared resources in deadline-driven environments.

9.2 Contributions

The conjunction of high-fidelity rendering and shared computational resources was mostly unexplored until now. This thesis built upon the existing knowledge in both these areas to develop new algorithms for operating in their conjunction. The major contributions of this thesis are:

- The use of a computational grid for rendering high-fidelity animations using a two-pass approach which gains speed-up and provides better visual quality when compared to traditional parallel animation rendering. (Chapter 5)
- A fault-tolerance mechanism for high-fidelity rendering using quasi-random sampling and image reconstruction techniques. This strategy does not inhibit performance unlike traditional fault-tolerance mechanisms. (Chapter 6, 7 and 8)
- The use of time-constraints for offline rendering on desktop grids. In addition, a component-based rendering approach which aims at maximising the visual quality of the renderings computed on variable resources within a user-specified time-constraint. (Chapter 6)
- A fully dynamic interactive high-fidelity rendering system capable of monitoring and adapting to run-time variation of resources while providing either a fixed frame rate or visual quality. (Chapter 7)
- A real-time spatio-temporal image reconstruction technique which can be used generating images from sparse sampling. (Chapter 7)
- The concept of time-constrained animation rendering in order to effectively employ desktop grids in a production environment. This is realised by using multi-dimensional quasi-random sampling which helps in progressively refining the whole animation even in the presence of faults. (Chapter 8)

9.3 Impact

The novel fault-tolerant mechanism developed in this thesis can be possibly used for various other applications such as data visualisation, computational fluid dynamics and other computer simulations where the results can be reconstructed from sparse samples. It would allow these applications to perform time-bounded

computations by adapting them at run-time on volatile resources without hindering performance.

The estimation of completion time of a computation is difficult when employing volatile resources, limiting their employment in a production environment where strict deadlines need to be adhered to. Hence, the concept of restricting the computation time, as presented in this thesis, to a user-defined time-constraint while maximising the output quality (visual quality in case of rendering), allows desktop grids to be effectively utilised in such environments.

Desktop grids have been traditionally utilised for high-throughput computing, however, this thesis demonstrates their capability of functioning as a high-performance resource for interactive rendering. Other research areas could also benefit from employing them for high-performance usage by adapting the computations in a similar fashion, instead of being restricted to expensive dedicated infrastructures.

9.4 Limitations and Directions for Future Work

Future work in the area of high-fidelity rendering on shared computational resources could address the restrictions of the presented algorithms. The two pass-algorithm presented in Section 5.2 has a bottleneck at the end of the first pass while merging the irradiance cache. It can be removed by distributing the merge process. Also, the second pass can be launched for the part of the animation as soon as the computation of irradiance values for that part is completed which would result in further speed-up. The current grid middleware for computational grids is not easily accessible for porting new applications. Hence, research needs to be done to reduce the learning curve for new users to make them pervasive.

The task scheduling in this thesis assumes equal importance for each pixel in the image/animation. This assumption can be lifted to further enhance the visual quality of the generated imagery by incorporating perceptual metrics such as saliency maps [IKN98]. These would allow the scheduler to prioritise visually important pixels. However, this would also add another level of complexity since they may have to be calculated and updated on the fly for optimum results. This may be especially limiting for rendering animations due to limited amount of memory as described in Section 8.5. Moreover, the application of such a technique would be even harder for interactive systems where the time-constraints are strict

and prioritising the pixels would require extra computations.

The current frameworks for computing on desktop grids are geared towards high-throughput computing and it takes a few minutes for them to transfer the files and start the job on idle machines. Future work would investigate ways for quick starting computations on desktop grids so that any ramp-up time can be minimised. Also, research needs to done to find better ways of synchronising the scene state on the workers for interactive rendering, so that it does not need to be communicated with each job packet.

There are many emerging distributed platforms such as cloud computing, autonomic computing and utility computing which can be employed for high-fidelity rendering by adapting the techniques for shared computing resources presented in this thesis. The current trend for increasing the computational power of a CPU is to increment the number of cores available on a single chip. The algorithms presented in this thesis can also be extended for employing such multi-core CPUs in a multi-programmed environment. Recently, the GPU is becoming increasingly flexible in terms of the computations that can be performed on it and this has been suitably exploited by the latest rendering algorithms [PBD*10]. Enhanced computational capabilities can be obtained by sharing multiple GPUs between users by employing techniques similar to the ones described here.

9.5 Final Remarks

The conjunction of rendering and shared resources presented in this thesis provides a glimpse of the possible capabilities, but many challenges still remain in order to fully harness the power of grid computing for rendering. The high cost of communication and the unpredictability of computing across the grid requires careful data management, load balancing and the design of fault-tolerant algorithms that can adapt to the resource availability. This thesis has been the first to investigate such algorithms. The described algorithms demonstrate effective utilisation of low cost variable resources, instead of traditional expensive parallel infrastructures, for both offline and interactive high-fidelity rendering. The work presented in this thesis has thus provided a firm foundation from which future research can build.

References

[ABD10] Adams A., Baek J., Davis M. A.: Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum 29*, 2 (2010), 753–762. 25

[ACD08] Aggarwal V., Chalmers A., Debattista K.: High-Fidelity Rendering of Animations on the Grid: A Case Study. In *Eurographics Symposium on Parallel Graphics and Visualization* (Crete, Greece, 2008), Eurographics Association, pp. 41–48. 58, 92

[ACK*02] Anderson D. P., Cobb J., Korpela E., Lebofsky M., Werthimer D.: Seti@home: an experiment in public-resource computing. *Communications of the ACM 45*, 11 (2002), 56–61. 33, 36

[ADBR*10] Aggarwal V., Debattista K., Bashford-Rogers T., Dubla P., Chalmers A.: High-fidelity interactive rendering on desktop grids. *IEEE Computer Graphics and Applications 99*, PrePrints (2010). 90

[ADD*09] Aggarwal V., Debattista K., Dubla P., Bashford-Rogers T., Chalmers A.: Time-constrained High-fidelity Rendering on Local Desktop Grids. In *Eurographics Symposium on Parallel Graphics and Visualization* (Munich, Germany, 2009), Eurographics Association, pp. 103–110. 70

[AFO05] Arikan O., Forsyth D. A., O'Brien J. F.: Fast and detailed approximate global illumination by irradiance decomposition. *ACM Transactions on Graphics 24*, 3 (2005). 22

[AGDL09] Adams A., Gelfand N., Dolson J., Levoy M.: Gaussian kd-trees for fast high-dimensional filtering. In *SIGGRAPH '09: ACM SIGGRAPH papers* (2009), ACM, pp. 1–12. 25

[AH93] Aupperle L., Hanrahan P.: A hierarchical illumination algorithm for surfaces with glossy reflection. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 155–162. 16

[AL81] Anderson T., Lee P. A.: *Fault Tolerance: Principles and Practice.* Prentice Hall, 1981. 39, 40

[ALR04] Avižienis A., Laprie J.-C., Randell B.: Dependability and its threats: A taxonomy. *Building the Information Society* (2004), 91–120. 39, 40

[AMHH09] Akenine-Möller T., Haines E., Hoffman N.: *Real-time rendering.* A K Peters Ltd., 2009. 15

[And04] Anderson D. P.: Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 4–10. 33, 37

[AOSM04] Al-Omari R., Somani A. K., Manimaran G.: Efficient overloading techniques for primary-backup scheduling in real-time systems. *Journal of Parallel and Distributed Computing 64*, 5 (2004), 629–648. 41

[App68] Appel A.: Some techniques for shading machine renderings of solids. In *AFIPS '68 (Spring): Proceedings of spring joint computer conference* (1968), ACM, pp. 37–45. 17

[AS79] Antonov I. A., Saleev V. M.: An economic method of computing lp[tau]-sequences. *USSR Computational Mathematics and Mathematical Physics 19*, 1 (1979), 252 – 256. 51

[ASKCK03] Antal G., Szirmay-Kalos L., Csonka F., Kelemen C.: Multiple strategy stochastic iteration for architectural walkthroughs. *Computers and Graphics 27*, 2 (2003), 285–292. 2

[Avi76] Avižiens A.: Fault-tolerant systems. *IEEE Transactions on Computers C-25*, 12 (1976), 1304 –1312. 39

[Ban06] Banino C.: Optimizing locationing of multiple masters for master-worker grid applications. *Applied Parallel Computing , Lecture Notes in Computer Science 3732* (2006), 1041–1050. 43

[Ban09] Bannore V.: *Iterative-Interpolation Super-Resolution Image Reconstruction: A Computationally Efficient Technique*. Springer Verlag, 2009. 23

[BBC*05] Brodlie K., Brooke J., Chen M., Chisnall D., Fewings A., Hughes C., John N. W., Jones M. W., Riding M., Roard N.: Visual supercomputing: Technologies, applications and challenges. *Computer Graphics Forum 24*, 2 (2005), 217–245. 47

[BF88] Bratley P., Fox B. L.: Algorithm 659: Implementing sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software 14*, 1 (1988), 88–100. 51

[BFGS86] Bergman L., Fuchs H., Grant E., Spach S.: Image rendering by adaptive refinement. *ACM SIGGRAPH Computer Graphics 20*, 4 (1986), 29–37. 26

[BFM10] Moscow State University Video Quality Tool/BFM Metric. `http://www.compression.ru/video/quality_measure/video_measurement_tool_en.html`, October 2010. 63

[BFMZ94] Bishop G., Fuchs H., McMillan L., Zagier E. J. S.: Frameless rendering: double buffering considered harmful. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), ACM, pp. 175–176. 25

[BHWL99] Bastos R., Hoff K., Wynn W., Lastra A.: Increased photorealism for interactive architectural walkthroughs. In *I3D '99: Proceedings of the symposium on Interactive 3D graphics* (1999), ACM, pp. 183–190. 2

[Boh05] Bohannon J.: DISTRIBUTED COMPUTING: Grassroots Supercomputing. *Science 308*, 5723 (2005), 810. 36

[BOI10] BOINC Project Statistics. `http://boincstats.com/stats/project_graph.php?pr=bo`, October 2010. 33, 37

[BP89] Badouel D., Priol T.: An efficient parallel ray tracing scheme for highly parallel architectures. In *Eurographics Hardware Workshop* (1989), Springer-Verlag. 28

[BWG03] Bala K., Walter B., Greenberg D. P.: Combining edges and points for interactive high-quality rendering. *ACM Transactions on Graphics 22*, 3 (2003), 631–640. 26

[BWS03] Benthin C., Wald I., Slusallek P.: A Scalable Approach to Interactive Global Illumination. *Computer Graphics Forum 22*, 3 (2003), 621–630. (Proceedings of Eurographics). 29, 91

[CAS86] Cristian F., Aghili H., Strong R.: Clock synchronization in the presence of omission and performance failures, and processor joins. In *Proceedings of 16th International Symposium on Fault-Tolerant Computing Systems* (1986), IEEE, pp. 218–223. 40

[CBH*04] Choi S., Baik M., Hwang C., Gil J., Yu H.: Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment. In *NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium* (2004), IEEE Computer Society, pp. 366–371. 38

[CBS*03] Cirne W., Brasileiro F., Sauve J., Andrade N., Paranhos D., Santos-Neto E., Medeiros R.: Grid computing for bag of tasks applications. In *Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment* (2003), Kluwer. 38

[CCEB03] Chien A., Calder B., Elbert S., Bhatia K.: Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing 63*, 5 (2003), 597–610. 39

[CDK01] Coulouris G., Dollimore J., Kindberg T.: *Distributed Systems, Concepts and Design.* Addison-Wesley, 2001. 41

[CDR02] Chalmers A., Davis T., Reinhard E. (Eds.): *Practical Parallel Rendering.* A. K. Peters, Ltd., 2002. 5, 28, 71

[Cho07] Choi S.: *Group-based Adaptive Scheduling Mechanism in Desktop Grid.* PhD Thesis, Korea University, 2007. 32, 43

[CL85] Chandy K. M., Lamport L.: Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems 3*, 1 (1985), 63–75. 42

[CLOB07] Chang M. W., Lindstrom W., Olson A. J., Belew R. K.: Analysis of HIV wild-type and mutant structures via in silico docking against diverse ligand libraries. *Journal of Chemical Information and Modeling 47*, 3 (2007), 1258–1262. 33, 36

[CON10] Condor project. `http://www.cs.wisc.edu/condor/`, October 2010. 38, 45

[CPC84] Cook R. L., Porter T., Carpenter L.: Distributed ray tracing. *SIGGRAPH Computer Graphics 18*, 3 (1984), 137–145. 18, 25

[CPD07] Chen J., Paris S., Durand F.: Real-time edge-aware image processing with the bilateral grid. In *SIGGRAPH '07: ACM SIGGRAPH papers* (2007), ACM, p. 103. 25

[Cro97] Crockett T. W.: An introduction to parallel rendering. *Parallel Computing 23*, 7 (1997), 819–843. 28

[CSL06] Chong A., Sourin A., Levinski K.: Grid-based computer animation rendering. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (2006), ACM, pp. 39–47. 46, 60, 92, 113

[CW93] Cohen M. F., Wallace J. R.: *Radiosity and Realistic Image Synthesis.* Morgan Kaufmann, 1993. 16

[Dal93] Daly S.: The visible differences predictor: an algorithm for the assessment of image fidelity. In *Digital images and human vision.* MIT Press, 1993, pp. 179–206. 83

[DBB06] Dutré P., Bala K., Bekaert P.: *Advanced Global Illumination.* A K Peters Ltd., 2006. 2, 10, 12, 16, 18, 19

[DCG*07] Debattista K., Chalmers A., Gillibrand R., Longhurst P., Mastoropoulou G., Sundstedt V.: Parallel selective rendering of high-fidelity virtual environments. *Parallel Computing 33*, 6 (2007), 361–376. 22

[Deb06] Debattista K.: *Selective Rendering for High-Fidelity Graphcs.* PhD Thesis, University of Bristol, 2006. 22, 27, 70

[Din99] Dinda P. A.: The statistical properties of host load. *Scientific Programming 7*, 3-4 (1999), 211–229. 38

[DMS05] Domingues P., Marques P., Silva L.: Resource usage of windows computer laboratories. In *2005 International Conference on Parallel Processing Workshops, Proceedings* (2005), pp. 469–476. 36, 38

[Dom08] Domingues P.: *Dependability Mechanisms for Desktop Grids.* PhD Thesis, University of Coimbra, 2008. 39

[Doo76] Dooley J. C.: Two-dimensional interpolation of irregularly spaced data using polynomial splines. *Physics of The Earth and Planetary Interiors 12*, 2-3 (1976), 180 – 187. 24

[DS84] Dippé M., Swensen J.: An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. *SIGGRAPH Computer Graphics 18*, 3 (1984), 149–158. 28

[DSC06] Debattista K., Santos L. P., Chalmers A.: Accelerating the irradiance cache through parallel component-based rendering. In *Eurographics Symposium on Parallel Graphics and Visualization* (Braga, Portugal, May 2006), Eurographics Association, pp. 27–34. 30

[DSMS07] Domingues P., Sousa B., Moura Silva L.: Sabotage-tolerance and trust management in desktop grid computing. *Future Generation Computer Systems 23*, 7 (2007), 904–912. 33, 41

[DSSC05] Debattista K., Sundstedt V., Santos L. P., Chalmers A.: Selective component-based rendering. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2005), ACM, pp. 13–22. 27, 78

[DWWL05] Dayal A., Woolley C., Watson B., Luebke D.: Adaptive frameless rendering. In *SIGGRAPH '05: ACM SIGGRAPH Courses* (2005), ACM, p. 24. 26

[EAWJ02] Elnozahy E. N. M., Alvisi L., Wang Y.-M., Johnson D. B.: A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys 34*, 3 (2002), 375–408. 42

[EGI10] Europian grid initiative. `http://www.egi.eu`, October 2010. 35

[ELvD*96] Epema D. H. J., Livny M., van Dantzig R., Evers X., Pruyne J.: A worldwide flock of condors: load sharing among workstation clusters. *Future Generation Computer Systems 12*, 1 (1996), 53–65. 46

[Fau82] Faure H.: Discrépance de suites associées à un système de numération (en dimension s). *Annals of the New York Academy of Sciences 41* (1982), 337–351. 49

[FDF03] Figueiredo R. J., Dinda P. A., Fortes J. A. B.: A case for grid computing on virtual machines. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems* (Washington, DC, USA, 2003), IEEE Computer Society, p. 550. 39

[FK99] Foster I., Kesselman C. (Eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 1999. 5, 33

[FK04] Foster I., Kesselman C. (Eds.): *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 2004. 34, 35

[FKNT02] Foster I., Kesselman C., Nick J., Tuecke S.: The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Services Infrastructure WG, Global Grid Forum* (2002). 43

[FKT01] Foster I., Kesselman C., Tuecke S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications 15*, 3 (2001), 200–222. 34

[Fos06] Foster I. T.: Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology 21*, 4 (2006), 513–520. 43, 44

[FS93] Funkhouser T. A., Séquin C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 247–254. 27

[FvDFH97] Foley J. D., van Dam A., Feiner S. K., Hughes J. F.: *Computer Graphics: Principles and Practice*. Addison Wesley, 1997. 15

[GAST06] Gooding S., Arns L., Smith P., Tillotson J.: Implementation of a distributed rendering environment for the teragrid. In *Challenges of Large Applications in Distributed Environments* (2006), IEEE, pp. 13–21. 47

[GB99] Gobbetti E., Bouvier E.: Time-critical multiresolution scene rendering. In *VIS '99: Proceedings of the conference on Visualization* (1999), IEEE Computer Society Press, pp. 123–130. 27

[GGHS03] Goesele M., Granier X., Heidrich W., Seidel H.-P.: Accurate light source acquisition and rendering. In *SIGGRAPH '03: ACM SIGGRAPH Papers* (2003), ACM, pp. 621–630. 3

[GIM10] Great Internet Mersenne Prime Search. `http://www.mersenne.org/`, October 2010. 36

[GKYL01] Goux J.-P., Kulkarni S., Yoder M., Linderoth J.: Master–worker: An enabling framework for applications on the computational grid. *Cluster Computing 4*, 1 (2001). 46, 81

[GLH*08] Gao J., Liu H., Huang J., Beck M., Wu Q., Moore T., Kohl J.: Time-Critical Distributed Visualization with Fault Tolerance. In *Eurographics Symposium on Parallel Graphics and Visualization* (Crete, Greece, 2008), Eurographics Association, pp. 65–72. 27, 47, 92

[GLO10] Globus Alliance. `http://www.globus.org/alliance/`, October 2010. 43

[GMWV*10] Gonzalez-Morcillo C., Weiss G., Vallejo D., Jimenez-Linares L., Castro-Schez J. J.: A multiagent architecture for 3d rendering optimization. *Applied Artificial Intelligence: An International Journal 24*, 4 (2010), 313–349. 47

[GN07] Garbacki P., Naik V. K.: Efficient resource virtualization and sharing strategies for heterogeneous grid environments. In *10th IFIP/IEEE International Symposium on Integrated Network Management* (2007), IEEE, pp. 40–49. 39

[GP90] Green S. A., Paddon D. J.: A highly flexible multiprocessor solution for ray tracing. *The Visual Computer 6* (1990), 62–73. 28

[GTGB84] Goral C. M., Torrance K. E., Greenberg D. P., Battaile B.: Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Computer Graphics 18*, 3 (1984), 213–222. 15

[HA98] Heirich A., Arvo J.: A competitive analysis of load balancing strategies for parallel ray tracing. *The Journal of Supercomputing 12*, 1-2 (1998), 57–68. 28

[Hal60] Halton J. H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik 2*, 1 (1960), 84–90. 49

[Ham60] Hammersley J.: Monte Carlo Methods for Solving Multivariable Problems. *Annals of the New York Academy of Sciences 86* (May 1960), 844–874. 49

[Hea03] Heap D. G.: *A Taxonomy of Actual Utilization of Real UNIX and Windows Servers.* White Paper GM12-0191, IBM, 2003. 36

[Her04] Hery C.: Rendering evolution at industrial light & magic. In *Proceedings of the 15th Eurographics Workshop on Rendering Techniques* (2004), Eurographics Association, pp. 19–22. 22

[HMD*10] Happa J., Mudge M., Debattista K., Artusi A., Gonçalves A., Chalmers A.: Illuminating the past: state of the art. *Virtual Reality 14* (2010), 155–182. 3

[ICG86] Immel D. S., Cohen M. F., Greenberg D. P.: A radiosity method for non-diffuse environments. *SIGGRAPH Computer Graphics 20*, 4 (1986), 133–142. 16

[IKN98] Itti L., Koch C., Niebur E.: A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20* (1998), 1254–1259. 133

[Jal94] Jalote P.: *Fault Tolerance in Distributed Systems.* Prentice Hall, 1994. 39, 40

[JDZJ07] Jarosz W., Donner C., Zwicker M., Jensen H. W.: Radiance caching for participating media. In *SIGGRAPH '07: ACM SIGGRAPH sketches* (2007), ACM, p. 56. 22

[Jen96] Jensen H. W.: Global illumination using photon maps. In *Proceedings of the Eurographics workshop on Rendering techniques* (1996), Springer-Verlag, pp. 21–30. 18

[Jen01] Jensen H. W.: *Realistic Image Synthesis Using Photon Mapping.* A K Peters Ltd., 2001. 12, 14

[Kaj86] Kajiya J. T.: The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), ACM, pp. 143–150. 2, 13, 18, 19, 20

[KBPv06] Křivánek J., Bouatouch K., Pattanaik S. N., Žára J.: Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. In *Rendering Techniques 2006, Eurographics Symposium*

on Rendering (Nicosia, Cyprus, June 2006), Eurographics Association, pp. 127–138. 22

[KE09] Kurt M., Edwards D.: A survey of brdf models for computer graphics. *SIGGRAPH Computer Graphics 43*, 2 (2009), 1–7. 13

[Kel97] Keller A.: Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 49–56. 18, 22

[KFC*07] Kondo D., Fedak G., Cappello F., Chien A. A., Casanova H.: Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems 23*, 7 (2007), 888–903. 36

[KGPB05] Křivánek J., Gautron P., Pattanaik S., Bouatouch K.: Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (September/October 2005). 22

[KH95] Keates M., Hubbold R.: Interactive ray tracing on a virtual shared-memory parallel computer. *Computer Graphics Forum 14*, 4 (1995), 189–202. 29

[KH01] Keller A., Heidrich W.: Interleaved Sampling. In *Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag. 80, 81

[KK02] Kollig T., Keller A.: Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3 (2002), 557–563. 57, 119

[KLC06] Keng S.-L., Lee W.-Y., Chuang J.-H.: An efficient caching-based rendering of translucent materials. *The Visual Computer 23*, 1 (2006), 59–69. 22

[KLN09] Kim S.-S., Lee J.-H., Nam S.-W.: Realistic rendering system using the measured brdfs. In *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences* (2009), ACM, pp. 1116–1121. 3

[KMG99] Koholka R., Mayer H., Goller A.: Mpi-parallelized radiance on sgi cow and smp. In *ParNum '99: Proceedings of the 4th International ACPC Conference* (1999), Springer-Verlag, pp. 549–558. 30

[KN74] Kuipers L., Niederreiter H.: *Uniform Distribution of Sequences*. Wiley-Interscience Publication, 1974. 48

[Knu81] KNUTH D. E.: *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Publication, 1981. 51

[Kon05] KONDO D.: *Scheduling Task Parallel Applications For Rapid Turnaround on Desktop Grids*. PhD Thesis, University of California, San Diego, 2005. 37, 38, 41

[LMG95] LONG D., MUIR A., GOLDING R.: A longitudinal survey of internet host reliability. In *SRDS '95: Proceedings of the 14TH Symposium on Reliable Distributed Systems* (1995), IEEE Computer Society, pp. 2–9. 38

[LS91] LIN T. T. Y., SLATER M.: Stochastic ray tracing using simd processor arrays. *Visual Computer 7*, 4 (1991), 187–199. 28

[LS98] LARSON G. W., SHAKESPEARE R.: *Rendering with Radiance: The Art and Science of Lighting Visualization*. Morgan Kaufmann Publishers Inc., 1998. 30, 59, 63

[LSS*04] LARSON S. M., SNOW C. D., SHIRTS M., P V. S., PANDE V. S.: Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In *Computational Genomics*. Horizon Press, 2004. 33, 36

[LTBL97] LITZKOW M., TANNENBAUM T., BASNEY J., LIVNY M.: *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*. Tech. Rep. 1346, University of Wisconsin-Madison Computer Science, 1997. 46

[LW93] LAFORTUNE E. P., WILLEMS Y. D.: Bi-directional path tracing. In *COMPUGRAPHICS 93: Proceedings of 3rd International Conference on Computational Graphics and Visualization Techniques* (1993), pp. 145–153. 18

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., 2002. 26

[MB97] MASON A. E. W., BLAKE E. H.: Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum 16*, 3 (1997). 27

[McN05] MCNAMARA A. M.: Exploring perceptual equivalence between real and simulated imagery. In *APGV '05: Proceedings of the 2nd symposium on*

Applied perception in graphics and visualization (2005), ACM, pp. 123–128. 63

[MDH07] Magoulès F., Diago L. A., Hagiwara I.: Efficient preconditioning for image reconstruction with radial basis functions. *Advances in Engineering Software 38*, 5 (2007), 320–32. 24

[Mel08] Meligy A.: Parallel and distributed visualization: The state of the art. In *CGIV '08: Proceedings of the Fifth International Conference on Computer Graphics, Imaging and Visualisation* (2008), IEEE Computer Society, pp. 329–336. 47

[Mes99] Messina P.: Distributed supercomputing applications. *The Grid: Blueprint for a New Computing Infrastructure* (1999), 55–73. 38

[Mit87] Mitchell D. P.: Generating antialiased images at low sampling densities. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM, pp. 65–72. 26

[Mit07] Mittring M.: Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH courses* (2007), ACM, pp. 97–121. 3

[ML91] Mutka M. W., Livny M.: The available capacity of a privately owned workstation environment. *Performance Evaluation 12*, 4 (1991), 269–284. 38

[MN98] Matsumoto M., Nishimura T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation 8*, 1 (1998), 3–30. 120

[MS95] Maciel P. W. C., Shirley P.: Visual navigation of large environments using textured clusters. In *I3D '95: Proceedings of the symposium on Interactive 3D graphics* (1995), ACM, pp. 95–102. 27

[Muu95] Muuss M. J.: Towards real-time ray-tracing of combinatorial solid geometric models. In *Proceedings of the Ballistic Research Laboratories Computer-Aided Design (BRL-CAD) Symposium* (1995). 29

[NGS10] National Grid Service, United Kingdom. `http://www.ngs.ac.uk/`, October 2010. 35, 44, 63

[Nic65] Nicodemus F. E.: Directional reflectance and emissivity of an opaque surface. *Applied Optics. 4*, 7 (1965), 767–773. 12

[Nie88] Niederreiter H.: Low-discrepancy and low-dispersion sequences. *Journal of Number Theory 30*, 1 (1988), 51–70. 49

[Nie03] Nielsen R. S.: *Real Time Rendering of Atmospheric Scattering Effects for Flight Simulators*. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2003. 4

[PB85] Plunkett D., Bailey M.: The vectorization of a ray-tracing algorithm for improved execution speed. *IEEE Computer Graphics and Applications 5*, 8 (1985), 52–60. 28

[PB89] Priol T., Bouatouch K.: Static load balancing for a parallel ray tracing on a mimd hypercube. *The Visual Computer 5* (1989), 109–119. 28

[PBD*10] Parker S. G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M.: Optix: a general purpose ray tracing engine. In *SIGGRAPH '10: ACM SIGGRAPH papers* (2010), ACM, pp. 1–13. 14, 110, 134

[PBMH02] Purcell T. J., Buck I., Mark W. R., Hanrahan P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3 (2002), 703–712. 29

[PFHA10] Pantaleoni J., Fascione L., Hill M., Aila T.: Pantaray: fast ray-traced occlusion caching of massive scenes. *ACM Transactions on Graphics 29*, 4 (2010), 1–10. 3

[PGAB*09] Patoli Z., Gkion M., Al-Barakati A., Zhang W., Newbury P., White M.: How to build an open source render farm based on desktop grid computing. In *Wireless Networks, Information Processing and Systems*, vol. 20 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2009, pp. 268–278. 47

[Pit93] Pitot P.: The voxar project (parallel ray-tracing). *IEEE Computer Graphics and Applications 13*, 1 (1993), 27–33. 28

[PKG*04] Pearlman L., Kesselman C., Gullapalli S., B. F. Spencer J., Futrelle J., Ricker K., Foster I., Hubbard P., Severance

C.: Distributed hybrid earthquake engineering experiments: Experiences with a ground-shaking grid application. *High Performance Distributed Computing* (2004), 14–23. 36, 44

[PMS*99] Parker S., Martin W., Sloan P.-P. J., Shirley P., Smits B., Hansen C.: Interactive ray tracing. In *I3D '99: Proceedings of the symposium on Interactive 3D graphics* (1999), ACM, pp. 119–126. 29

[PPL*99] Parker S., Parker M., Livnat Y., Sloan P.-P., Hansen C., Shirley P.: Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 238 –250. 29

[PS89] Painter J., Sloan K.: Antialiased ray tracing by adaptive progressive refinement. *ACM SIGGRAPH Computer Graphics 23*, 3 (1989), 281–288. 26

[PSL*98] Parker S., Shirley P., Livnat Y., Hansen C., Sloan P.-P.: Interactive ray tracing for isosurface rendering. In *VIS'98: Proceedings of the conference on Visualization* (1998), IEEE Computer Society Press, pp. 233–238. 29

[PTVF07] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007. 48

[RCLL99] Robertson D., Campbell K., Lau S., Ligocki T.: Parallelization of radiance for real time interactive lighting visualization walkthroughs. In *Supercomputing '99: Proceedings of the ACM/IEEE conference on Supercomputing* (1999), ACM, p. 61. 30

[RFWB07] Ramanarayanan G., Ferwerda J., Walter B., Bala K.: Visual equivalence: towards a new standard for image fidelity. In *SIGGRAPH '07: ACM SIGGRAPH papers* (2007), ACM. 83

[RGS00] Reisman A., Gotsman C., Schuster A.: Interactive-rate animation generation by parallel progressive ray-tracing on distributed-memory machines. *Journal of Parallel and Distributed Computing 60*, 9 (2000), 1074–1102. 27, 29

[RLS98] Raman R., Livny M., Solomon M.: Matchmaking: Distributed resource management for high throughput computing. In *HPDC '98: Proceedings of the 7th IEEE International Symposium on High Performance*

Distributed Computing (Washington, DC, USA, 1998), IEEE Computer Society, p. 140. 45

[RMS*08] Rump M., Muller G., Sarlette R., Koch D., Klein R.: Photo-realistic rendering of metallic car paint from image-based measurements. *Computer Graphics Forum 27* (2008), 527–536. 3

[SaLY*08] Sitthi-amorn P., Lawrence J., Yang L., Sander P. V., Nehab D.: An improved shading cache for modern gpus. In *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2008), Eurographics Association, pp. 95–101. 26

[SAWG91] Sillion F. X., Arvo J. R., Westin S. H., Greenberg D. P.: A global illumination solution for general reflectance distributions. *SIGGRAPH Computer Graphics 25*, 4 (1991), 187–196. 16

[SC88] Scherson I. D., Caspary E.: Multiprocessing for ray tracing: a hierarchical self-balancing approach. *The Visual Computer 4* (1988), 188–196. 28

[SEB08] Shirley P., Edwards D., Boulos S.: Monte carlo and quasi-monte carlo methods for computer graphics. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer Berlin Heidelberg, 2008, pp. 167–177. 23

[SG88] Salmon J., Goldsmith J.: A hypercube ray-tracer. In *C3P: Proceedings of the third conference on Hypercube concurrent computers and applications* (1988), ACM, pp. 1194–1206. 28

[SH82] Shoch J. F., Hupp J. A.: The "worm" programs—early experience with a distributed computation. *Communications of the ACM 25*, 3 (1982), 172–180. 36

[She68] Shepard D.: A two-dimensional interpolation function for irregularly-spaced data. In *ACM '68: Proceedings of the 23rd ACM national conference* (1968), ACM, pp. 517–524. 23

[SKDM05] Smyk M., Kinuwaki S., Durikovic R., Myszkowski K.: Temporally coherent irradiance caching for high quality animation rendering. In *Proceedings of Eurographics* (2005), vol. 24, Blackwell, pp. 401–412. 22

[SM09] Shirley P., Marschner S.: *Fundamentals of Computer Graphics*. A K Peters Ltd., 2009. 12, 14, 16, 17

[Sob67] Sobol I. M.: On the distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Computational Mathematics and Mathematical Physics 7* (1967), 86–112. 49, 51, 54

[SP89] Sillion F., Puech C.: A general two-pass method integrating specular and diffuse reflection. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (1989), ACM, pp. 335–344. 16

[SS98] Silva L. M., Silva J. G.: System-level versus user-defined checkpointing. In *SRDS '98: Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems* (Washington, DC, USA, 1998), IEEE Computer Society, p. 68. 41

[SS00] Simmons M., Séquin C. H.: Tapestry: A dynamic mesh-based display representation for interactive rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques* (2000), Springer-Verlag, pp. 329–340. 26

[SS08] Schurer R., Schmid W. C.: van der Corput Sequence. *From MinT–the database of optimal net, code, OA, and OOA parameters. Version: 2008-04-04* (2008). `http://mint.sbg.ac.at/desc_SCorput.html`. 50

[SSH*98] Slusallek P., Stamminger M., Heidrich W., Popp J.-C., Seidel H.-P.: Composite lighting simulations with lighting networks. *IEEE Computer Graphics and Applications 18*, 2 (1998), 22–31. 78

[TACBI05] Taufer M., Anderson D., Cicotti P., Brooks III C. L.: Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium - Workshop 1* (2005), IEEE Computer Society, p. 119.1. 39

[TER10] Teragrid. `https://www.teragrid.org/`, October 2010. 35, 44

[TL04] Tabellion E., Lamorlette A.: An approximate global illumination system for computer generated films. *ACM Transactions on Graphics 23*, 3 (2004), 469–476. 3, 22

[TM98] Tomasi C., Manduchi R.: Bilateral filtering for gray and color images. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision* (1998), IEEE Computer Society, p. 839. 24

[TMD*04] Tawara T., Myszkowski K., Dmitriev K., Havran V., Damez C., Seidel H.-P.: Exploiting temporal coherence in global illumination. In *SCCG '04: Proceedings of the 20th Spring Conference on Computer graphics* (2004), ACM Press, pp. 23–33. 22

[TOP10] TOP500 Project. `http://www.top500.org/lists/2010/06`, October 2010. 37

[TPWG02] Tole P., Pellacini F., Walter B., Greenberg D. P.: Interactive global illumination in dynamic scenes. *ACM Transactions on Graphics 21*, 3 (2002), 537–546. 26

[VALBW06] Velázquez-Armendáriz E., Lee E., Bala K., Walter B.: Implementing the render cache and the edge-and-point image on graphics hardware. In *GI'06: Proceedings of Graphics Interface* (2006), Canadian Information Processing Society, pp. 211–217. 26

[vdC35] van der Corput J. G.: Verteilungsfunktionen I and II. In *Proc. Nederl. Akad. Wetensch.* (1935). 49, 50

[VG97] Veach E., Guibas L. J.: Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 65–76. 18

[WDG02] Walter B., Drettakis G., Greenberg D. P.: Enhancing and optimizing the render cache. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 37–42. 26

[WDP99] Walter B., Drettakis G., Parker S.: Interactive rendering using the render cache. In *Rendering techniques '99: Proceedings of the 10th Eurographics Workshop on Rendering* (June 1999), vol. 10, Springer-Verlag/Wien, pp. 235–246. 25

[WH92] Ward G., Heckbert P.: Irradiance gradients. In *Third Annual Eurographics Workshop on Rendering* (1992), Springer-Verlag, pp. 85–98. 21

[Whi80] Whitted T.: An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (1980), 343–349. 17

[WKB*02] Wald I., Kollig T., Benthin C., Keller A., Slusallek P.: Interactive global illumination using fast ray tracing. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 15–24. 18, 22, 29, 91

[WLC10] Worldwide lhc computing grid. `http://lcg.web.cern.ch/LCG/public/`, October 2010. 35, 44

[WLW02] Woolley J. C., Luebke D., Watson B.: Interruptible rendering. In *SIGGRAPH '02: ACM SIGGRAPH conference abstracts and applications* (2002), ACM, pp. 205–205. 26

[Woo84] Woodwark J.: A multiprocessor architecture for viewing solid models. *Displays 5*, 2 (1984), 97 – 103. 28

[WRC88] Ward G. J., Rubinstein F. M., Clear R. D.: A ray tracing solution for diffuse interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), ACM, pp. 85–92. 18, 21

[WS99] Ward G., Simmons M.: The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Transactions on Graphics 18*, 4 (1999), 361–368. 26

[WSBW01] Wald I., Slusallek P., Benthin C., Wagner M.: Interactive rendering with coherent ray tracing. In *Eurographics 2001 Proceedings*, vol. 20(3). Blackwell Publishing, 2001, pp. 153–164. 29

[WSH00] Wolski R., Spring N., Hayes J.: Predicting the cpu availability of time-shared unix systems on the computational grid. *Cluster Computing 3*, 4 (2000), 293–301. 38

[YPG01] Yee H., Pattanaik S., Greenberg D. P.: Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics 20*, 1 (2001), 39–65. 22

[YPZ10] Yao J., Pan Z., Zhang H.: A distributed render farm system for animation production. In *Entertainment Computing – ICEC 2009*, vol. 5709. Springer Berlin / Heidelberg, 2010, pp. 264–269. 30

[ZA10] Zhu Q., Agrawal G.: Supporting fault-tolerance for time-critical events in distributed environments. *Scientific Programming 18*, 1 (2010), 51–76. 47

[ZMK02] Zach C., Mantler S., Karner K.: Time-critical rendering of discrete and continuous levels of detail. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology* (2002), ACM, pp. 1–8. 27